SPECIAL PURPOSE PROCESSORS

C. Verkerk
CERN, Geneva, Switzerland

INTRODUCTION

The subject of these lectures will be special purpose processors implemented in hardware. The subject matter is difficult to treat in a general way since due to their specialisation it is only practical to present specific examples of processors. At best one can hope to distinguish some classes of processors, where the classes correspond in general to the application area, rather than to the architecture or design philosophy. Different pattern recognition and picture processing machines have points in common and Fast Fourier Transform processors bear a ressemblance to each other. Indeed when one takes a closer look at the different special purpose machines proposed or built, one is struck by the very close relation between the hardware and the algorithm the machine has been built to execute. This could possibly be transposed at least for the purpose of the present lectures into a definition: a special purpose processor is a piece of hardware designed to fit as closely as possible the algorithm it is built to execute.

The point is that the hardware has been designed for the algorithm [1] and that no programming has been done to implement it on a machine which can be reprogrammed. A close relation exists between the algorithm and its implementation in special purpose hardware.

The incapacity of being programmed in the usual sense of the word does not necessarily mean that special hardware is completely rigid and unadaptable. Of course, it would be unthinkable to adapt a Fast Fourier Transform processor for use in, say, a numerical control application. But the FFT processor can be used for many tasks (transform, convolution, correlation, digital filtering). Often the change from one task to another can be made at the flick of a switch. The definition would exclude special purpose systems which are based on a single microprocessor (computer on a chip, or rather a handful of chips) or a minicomputer, microprogrammable or not. Neither are larger special systems, implemented around a general purpose machine a subject for these lectures. Parts of such a system could well fit our definition (e.g. I/O channels, or the special processor for filtering slice scan data in the Erasme system [58]).

In agreement with the -rather restrictive- definition of special processors given above, the present lectures will try first to develop some general points on the applications and motivations of hardware processors. More specific points will be discussed in the examples which will be covered in some detail : Fast Fourier Transform Processors and hardware for track recognition in high-energy physics experiments. The emphasis will be on processors for wire chamber data, but a rapid survey will be made of special hardware for filtering of digitized coordinates from bubble chamber film.

Why and When Special Purpose Processors?

Let us try to answer the question "why special purpose processors?" The answer is already contained to some extent in what was said before: for some problems programming a general purpose computer is not necessarily the optimum solution. Special hardware can be more cost-effective than the running of a programme on a general purpose machine or - if it is not - it can overcome a real-time constraint. The FFT hardware provides an example for both motivations: it was soon realized that FFT processor would be very cost-effective [2] and that they would allow real-time analysis of Doppler Radar returns [3], [4] or of human speech [5]. Picture processing (image restoration, image enhancement, difference detection) and pattern recognition in photographic images also provide strong motivations for studying special hardware solutions. The reason, of course, is that the large number of picture cells to be handled, combined with grey-level and colour information make the processing task formidable and time-consuming on a sequential machine. The tendency in this field is to think in terms of highly parallel structures (cellular automata). References [6] to [10], far from being exhaustive give some examples. An exception to this cellular approach seems to be the Control Data picture processing machine [11].

Both motivations also exist in high-energy physics. Many analysis programs spend a large fraction of time in a few inner loops. The calculations in these loops are often very simple and it can be cost-effective to perform them outside the large computer. The data presented for further analysis have then already been filtered elsewhere. The real-time constraint can be present when a selection of events has to be made at an early stage (i.e. before recording on magnetic tape, or before triggering the flash lights of a rapid cycling bubble chamber [57]).

Some of the answers to the next question: "what kind of problems are good candidates for implementation in special hardware?" have become apparent already.

i) Obviously the first condition is that there be a sufficiently continuous stream of data to analyse, in order to give a reasonable life-span to the processor. In other words: no special hardware for a one-off problem.

ii) Secondly the algorithm should be to a large extent iterative or repetitive, which implies that some kind of a loop-structure should be present. Nested loops are ideal, because a large part of the overheads normally present in the loop control can be eliminated.

iii) The simpler the arithmetic or logical operations to be performed repetitively, the more suitable the algorithm is for hardware implementation.

iv) The required precision should be rather small, so that 12 or 16-bit fixed point arithmetic may be used. Floating point operations on longer words are not excluded, but the cost of the hardware would increase sharply.

v) A simple control structure will result in simple hardware. A straightforward algorithm, where no special cases have to be considered, would be ideal. This point should not be exaggerated though. We will see later that IF-statements, conditional jumps and the like can be implemented, often more easily and more efficiently than in a program. The question of control is also closely linked to the problem of flexibility, to which we will come back.

vi) Finally, a last asset for hardware implementation of an algorithm is that it consists of independent parts. When different calculations are independent of each other, they can be performed in parallel. Another possibility is that calculations can be performed in a pipeline. This is the case when later steps in the algorithm do not make use of hardware necessary for the execution of earlier steps. A continuous stream of "events" can then be maintained, the different events in the pipeline residing in different sections of hardware and being in different stages of progress.

Influence of Technology

It is commonplace to say that the extremely fast development of the integrated circuit technology over the last ten years has brought about a revolution in digital design. I apologise for repeating it, but it is worthwhile to point out that the present variety, availability and price of integrated circuits has brought it within the possibilities of individuals in a laboratory to do things they would not have dreamt of five years ago. This is particularly true since MSI and LSI circuits are readily available. This has really brought it within the reach of many an electronic designer to build special purpose systems. Until recently his designs were often limited to data acquisition and interfacing problems, without arithmetic capabilities. To open the way to the design of special purpose processors, it is sufficient to become aware of the possibilities offered by including arithmetic. By studying carefully the algorithm he is to implement he will be able to find elegant solutions which speed up execution or reduce the amount of hardware. The chances are that the average designer will find original solutions, if he works towards the closest fit between the hardware and the algorithm. If not, and if he remembers too much von Neumann, the result will probably be another item on the already long list of microprogrammed or microprogrammable minicomputers.

It is probably worth mentioning that the full impact of the integrated circuit revolution on the computer industry is still not felt. There are some signs of it coming [12]. Significant are the laments that computer architects are not sufficiently aware of the present hardware possibilities. These laments are accompanied by recommendations to stimulate hardware research in American Universities [13]. This prospect of entrusting more and more system tasks (processes) to hardware processors in the framework of general purpose computers constitutes in fact the bridge between the present lectures and the other lectures on computer architecture. Conceptually, very little would change for the systems architect if, for instance, a loader would be implemented in hardware or in software. The position taken in the present lectures contrasts however with the general trend in systems programming on one point: binding. Whereas the systems programmer delays binding to the ultimate moment, this is not the case - yet - in special purpose hardware. There every datum is fixed to its place right from the beginning.

This early binding and the fact that the hardware is made to fit the algorithm as a glove relieves us from many of the problems encountered in general purpose systems: e.g. fast operand fetching, adaption of programs to multiprocessor systems, optimum use of arithmetic and other resources. Early binding should not be confused with lack of flexibility! We will come back on the presumed inflexibility of hardware in the examples we will treat in detail.

FAST FOURIER TRANSFORM PROCESSORS

A vast amount of literature exists on the FFT (see ref. [2] for an extensive list of earlier references). We will therefore not dwell upon details, limitations, pitfalls, etc., but limit ourselves to one specific example of a hardware implementation [3] which illustrates best the points we want to make.

The Algorithm

The discrete version of the Fourier transform, applicable to time samples of a continuous function x(t) is:

$$X(j) = \frac{1}{N} \sum_{j=0}^{N-1} x(k) e^{-i.2\pi jk/N} \qquad (1)$$

with its inverse:

$$x(k) = \sum_{j=0}^{N-1} X(j) e^{\,i.2\pi jk/N} \qquad (2)$$

both for $j = 0,1....,\ N-1$ and $k = 0,1,....,N-1$

Both the spectrum X(j) and the time series x(k) are, in general, complex series. Replacing $e^{2\pi i/N}$ by W, we can write the essential part of (1) or (2) as:

$$X(j) = \sum_{k=0}^{N-1} A(k) W^{jk} \qquad (3)$$

For the sake of clarity we will limit ourselves to the case N=8. We will then write j and k as binary numbers : $j=j_2.4 + j_1.2 + j_0$ and $k=k_2.4 + k_1.2 + k_0$.

Eq. (3) then becomes:

$$X(j_2,j_1,j_0) = \sum_{k_0=0}^{1} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} A(k_2,k_1,k_0) W^{(j_2.4+j_1.2+j_0)(k_2.4+k_1.2+k_0)}$$

Splitting up the exponent into three partial products and remembering that $W^8=1$ we obtain:

$$X(j_2,j_1,j_0) = \sum_{k_0=0}^{1} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} A(k_2,k_1,k_0) W^{j_0 k_2.4}\, W^{(j_1.2+j_0)k_1.2}\, W^{(j_2.4+j_1.2+j_0)k_0}$$

This can be calculated in three steps:

$$A_1(j_0,k_1,k_0) = \sum_{k_2=0}^{1} A(k_2,k_1,k_0)W^{j_0k_24} \qquad (4)$$

$$A_2(j_0,j_1,k_0) = \sum_{k_1=0}^{1} A_1(j_0,k_1,k_0)W^{(j_12+j_0)k_12} \qquad (5)$$

$$A_3(j_0,j_1,j_2) = \sum_{k_0=0}^{1} A_2(j_0,j_1,k_0)W^{(j_24+j_12+j_0)k_0} \qquad (6)$$

with final transposition:

$$X(j_2,j_1,j_0) = A_3(j_0,j_1,j_2) \qquad (7)$$

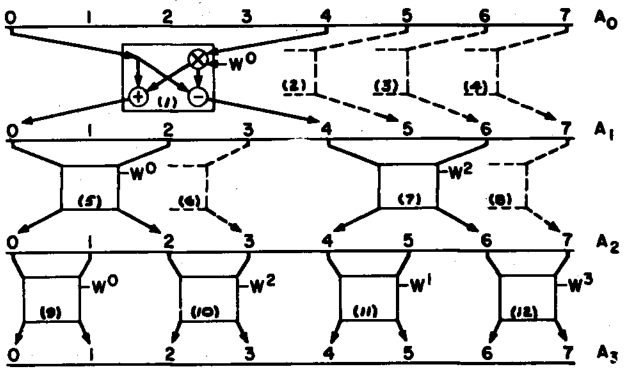


Figure 1 - FTT Data Flow for N=8

Note that each step calculates 8 new terms from the old ones. Only the results of step m are needed in step (m+1) and the older results can be discarded. The importance of the FFT lies in the fact that the N^2 complex operations (one multiplication and one addition) to calculate (3) are replaced by $N\log_2 N$ operations. By noting that $W^{\ell} = -W^{(\ell+N/2)}$ we see that we have to perform only $\frac{N}{2}\log_2 N$ times the following calculations:

$$x_{out} = x_{in} + y_{in}.W^z \qquad (8)$$

$$y_{out} = x_{in} - y_{in}.W^z \qquad (9)$$

(or, in the previous notation, for instance:

$$A_2(5) = A_1(5) + A_1(7)W^2$$

$$A_2(7) = A_1(5) - A_1(7)W^2 \quad)$$

The process, still for N=8 is illustrated in Figure 1 (from [14]). Obviously the general algorithm [15] is not limited to N=8, nor even to $N=2^m$, although most hardware implementations have the last restriction. For earlier hardware implementations see Bergland [16] .

Hardware Implementation

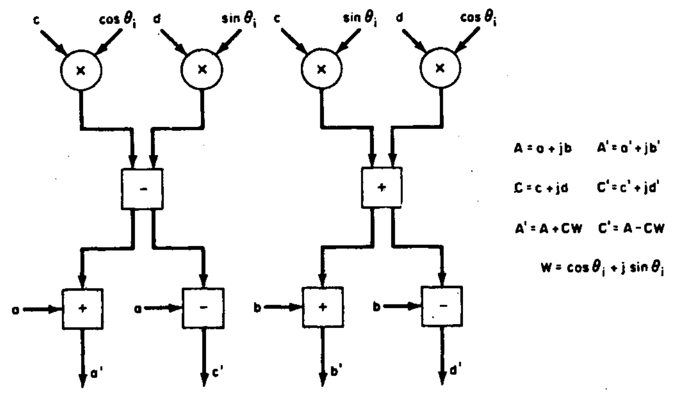

c
$\cos\theta_i$
d
$\sin\theta_i$
c
$\sin\theta_i$
d
$\cos\theta_i$
×
×
×
×
-
+
a
+
a
-
b
+
b
-
a'
c'
b'
d'
A = a + jb   A' = a' + jb'
C = c + jd   C' = c' + jd'
A' = A + CW   C' = A - CW
W = $\cos\theta_i + j\sin\theta_i$

Figure 2 - Arithmetic Element for FFT

The heart of a FFT processor is the arithmetic module to perform the complex calculations (8) and (9). In terms of real numbers, this "butterfly" can be implemented as shown in Figure 2. It is important to realize that this implementation constitutes a big advantage over the general purpose approach where - generally speaking - only a single arithmetic operation would be in progress at any instant in time. Note that a further speed-up could be obtained by organising the butterfly in a pipeline : a new multiplication could start as soon as the past one is ready and before the results of the additions and substractions are known. This would lead, however, to complications in addressing the data. Other, cheaper designs of the butterfly are possible, requiring less multipliers and adders [17] , but they are slower, due to their sequential operation.

Bergland [14] distinguishes four organisations of the FFT processor:

i) Sequential. A single arithmetic module (butterfly) performs all operations sequentially in a total time $T.\frac{N}{2}\log_2 N$. [5], [18] , [19] , [24] .

ii) Cascade. There is one butterfly for each iteration. This requires $\log_2 N$ butterflies, each performing N/2 operations sequentially. It will still take $T.\frac{N}{2}\log_2 N$ before the spectrum of a data set is available, but the throughput is increased by a factor $\log_2 N$, since $\log_2 N$ data sets can be simultaneously in the pipeline [2], [3].

iii) Parallel iterative. Here the parallelism is in the horizontal direction in Figure 1. So there are N/2 processing elements and the execution time is $T\log_2 N$. [21] , [22] .

iv) Array. Parallelism is extended in both directions. $\frac{1}{2}N\log_2 N$ butterflies are needed, the throughput is one data set every T seconds [23] .

It is clear that the last two organisations are rather unrealistic for data sets of 1024 or more samples. In what follows we will therefore give as an example a cascade (or pipeline) implementation, which is the one built by Groginsky and Works [3].

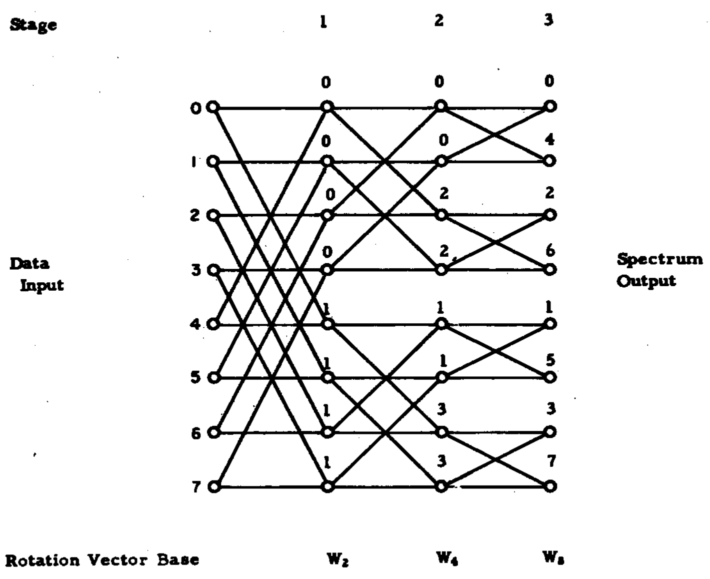


Figure 3 - Flow Diagram of FTT

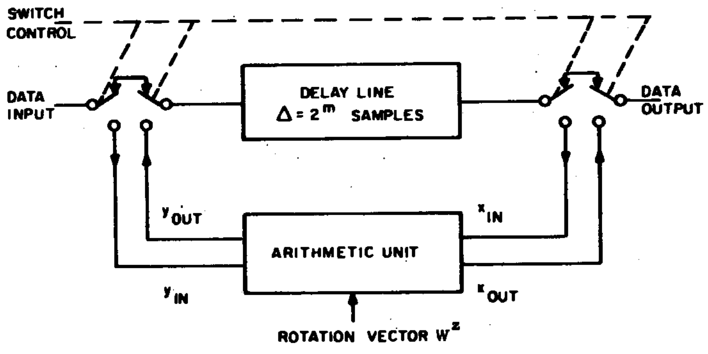


Figure 4 - Module m of Pipeline FFT

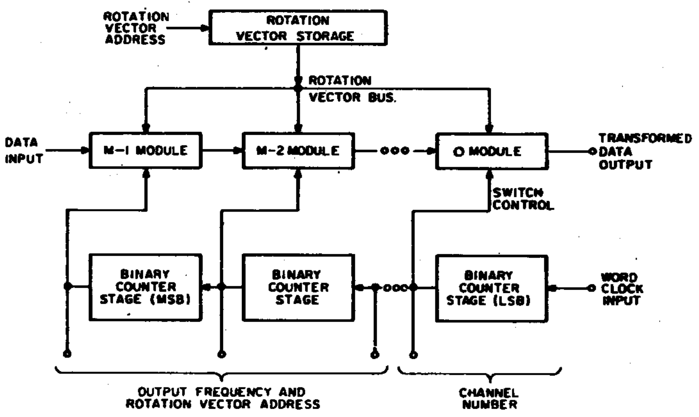


Figure 5 - Block Diagram of FFT Processor

Let us first have a look at Figure 3, which is another drawing of the flow of data in the FFT algorithm. We can then note the following:

i) each iteration needs only the results of the preceding stage.

ii) the first iteration uses data samples separated by a distance N/2, the second by N/4, etc.

iii) the stages are independent in the sense that one stage can be working on data from one source, while another stage treats data from another source (storage must be provided between stages).

iv) the rotation vector required at each stage has the same periodicity as the displacement between data samples.

These considerations lead to the implementation of stage m which is shown in Figure 4. The delay line (actually a shift register) can contain 2^m samples (total delay time $\Delta=2^m. \delta$ when δ=sampling time interval). The arithmetic unit performs calculations (8) and (9). The switches are controlled as follows: first 2^m data samples are stored in the delay line and when it is full a rotation vector W^z is obtained and the switches are thrown. So the arithmetic module now starts working, receiving one input (x_{in}) per sampling interval δ from the delay line, the other input (y_{in}) directly from the data source. One output (x_{out}) is passed on immediately to the next stage, to be stored in the delay line there (the latter can contain only 2^{m-1} samples). The other output is fed back into the local delay line, so that it will become available to the next stage after 2^m sampling intervals. This is just what is needed (see Figure 3). In stage m the delay line is thus alternatively filled with fresh data and with results, in blocks of 2^m. The next stage, working on blocks of 2^{m-1} data needs to throw switches twice as often as stage m. The control of the switches can therefore be performed with the help of a binary counter! The total organisation then becomes as in Figure 5.

The rotation vectors are stored in a (read-only) memory in the order required for the last iteration. The binary counter provides the addresses for this memory. The stages upstream, which need a lower periodicity for the rotation vectors, strobe the correct value into an internal register at the moment its switches are thrown.

There is still another trick present in this implementation. As can be seen from Figure 3, when two independent data streams are interleaved (even and odd samples belong to different data sets) then the spectral components of both sets are available at the intermediate output of stage 1, interleaved also. In general, for 2^k interleaved sets, the interleaved spectral components (in the same channel order) are available at the output of module k. So one can easily choose between a single stream of 1024 samples, or 4 streams of 256 samples each, etc.

This processor has in fact been built and it processes eight range channels of a Doppler radar, taking 512 complex samples per channel. The throughput rate achieved is 128K samples per second. The processor uses 12-bit fixed point arithmetic.

As with most other processors, the spectral components within a channel are produced in scrambled (bit-reversed) order, which is inherent to the Cooley-Tukey algorithm (see Figure 3 and equation 7).

This example illustrates very well the point made before: by looking closely at the algorithm an elegant and fast processor has been produced.

SPECIAL PROCESSORS FOR HIGH-ENERGY PHYSICS

Bubble Chambers

Apart from the one already mentioned [7], other proposals [25 - 35] have been made to perform track-element search by special hardware and a number of projects [27 - 30 - 32 - 35] have been launched in the past to replace by hardware at least a major part of the track-following programs for data from flying spot digitizers. Most of these hardware approaches (LER, [25 - 26], BRUSH, [27 - 29] , PANGLOSS, [34]) are based on histogramming technique. A

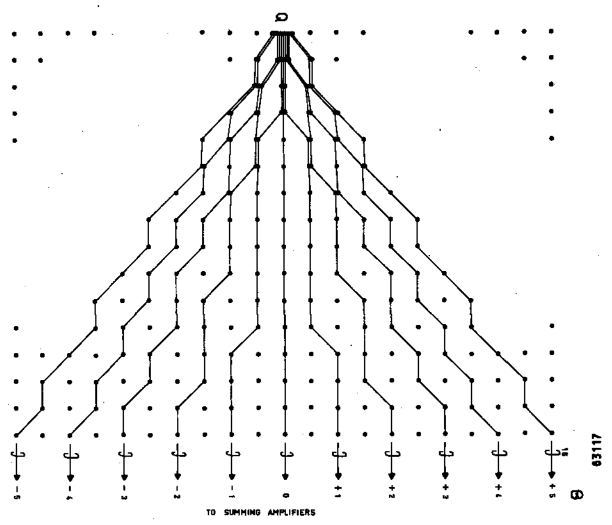


Figure 6 - Pattern for Line Element Search

histogramming technique which does not use arithmetic, but only logic. The hits of the flying spot on track images are not represented numerically as the value of the coordinate, across the film, but rather as bits in a shift register. The bit pattern in the shift register is a representation of the analogue signal produced by the photo-detector. A number of scans is represented in an array of registers. The shift register array (which is shorter than corresponds to a complete scan-line, scan-lines being shifted through it) contains pre-wired patterns, approximating straight-line segments under different angles (see Figure 6). At every shift a hit count is established for each direction. A maximum must be sought in successive hit counts (always for each of the fixed directions) and a decision taken on the possible presence or absence of a line-element. The shift register array contains only a limited number of scan-lines (slice). Next the line-elements found in successive slices must be linked into tracks, which is another non-trivial problem. BRUSH [28] has in fact been implemented, and is undergoing extensive tests.

COCCINELLE [32 - 33] followed another approach, purely analogue. A very strict synchronisation between line sweeps is imposed. A line element under a given angle is then characterized by the recurrence of a fixed time delay between successive signals. A set of delay lines is used to detect recurrent pulses. Output pulses are fed back into the same delay line, after being added to the input signal present at this particular instant in time. Pulses separated by the delay time build up to large amplitudes, dominating a sea of small pulses.

SATR [30 - 31] adapted a three-dimensional approach. It needs precise knowledge of the fiducial positions before processing can start. Every hit on the four stereo-views is transformed into a light ray in space. Small regions of space are then searched for close crossings of light rays. A track element detected is followed in three dimensions.

It is not known to the author if SATR reached a full scale implementation.

A less ambituous approach was taken at the Zeeman Laboratory in Amsterdam, where special hardware was built to follow beam tracks only [35]. A histogramming technique is followed, but using the numerical coordinate values. The hardware is thus a close replica of the kernel of a track-following program. Moreover, approximate position, incident angle and curvature are known for beam tracks. This approach seems to be successful and capable of following beam-tracks and detecting their disappearance at an interaction vertex. Finally, one should mention a special purpose processor under development for ERAMSE at CERN [36]. Here the aim is to speed up the analysis, decentralising still further the different tasks [58]. The analysis of a slice scan to find the parameters of a track segment will thus be done in specialised hardware, which has taken the form of a micro-programmed processor with a specialised instruction set.

Wire Chambers

For the treatment of wire chamber data a number of special processors have been built or proposed. The first to be mentioned (although chronologically one of the last) is MEDEA [37], a special processor to separate a continuous stream of coordinates into different sets, each set originating from a different wire plane. In addition, clusters of "sparks" are detected and transformed into a coordinate corresponding to the center of the cluster. The wire planes can have an arbitrary number of wires. Note that this processor could have been avoided with another design of the read-out system.

Three groups (not counting the detailed example to be given later) tackled the problem of detecting straight particle tracks in wire chamber set-ups.

The first one [38] uses a purely arithmetic method to detect if in a single particle event the points in 3 detectors lie on a straight line. If these detectors are indicated by A, B and D respectively, then the relation checked is:

$$|X_A-X_B+X_D-C_X| < \varepsilon \qquad (10)$$

and similarly for the Y-coordinates. C_X and C_Y are constants. Three adders and a comparator are used to check (10) for one coordinate. When (10) is satisfied for both X and Y, the event is rejected, since the experiment is intended to study elastic scattering and the chambers A and B are in front of the target and chamber D behind it. This hardware has been effectively used in an experiment.

The second approach [39, 40] is very similar to a number of the bubble chamber approaches (BRUSH, LER): the "sparks" for each detector are stored as bits in shift registers, the position of the bit being an image of the position of the spark. Instead of using pre-wired patterns to search for coincidences, the different angles are scanned by first off-setting the shift registers by the amount required for the search angle. The complete shift register array is then shifted along a single coincidence unit. By ORing successive bits in this coincidence unit, the resolution can be varied. The method is not very fast: for four 128-wire proportional chambers, 1024 4-bit wide searches must be made, taking a time of 100 μs. Adding I/O time and the time for high-resolution searches one arrives at $\approx$600 μs per view. This time only depends linearly on the number of tracks.

The third processor is more sophisticated [41]. First it finds in a set-up of 4 parallel wire chambers all combinations of 3 points on a straight line. This is done in the "Intelligent Memory", for a maximum of 16 particles. The IM has 256 locations where first all possible values of $C_{1i}X_i+C_{3k}X_k$ are stored for i=1,...,16 and k=1,...,16. These values are then compared with the values of $C_{2j}X_j$ (for j=1,...,16). When a match is found, the three points X_i,X_j and X_k in chamber 1, 2 and 3 lie on a straight line. A bit is then stored in another memory C, at an address which is directly related to the address in the Intelligent Memory where the match was found. This address is in fact a pointer to the combination of points on the line. The process is repeated for chambers 1, 2 and 4, then for 1, 3 and 4 and finally for 2, 3 and 4. A cleaning-up process is then performed to combine for each particle the four lines through 3 points into a single one through 4 points. The approach seems neat and able to deal easily with inefficiencies.

Lastly, and before we leave this survey to treat in detail a hardware processor which is able to deal with curved tracks as well, it is worth mentioning the polynomial generator of McPherson and Wilde [42]. This processor can evaluate other formulae as well, and its application is not limited to high-energy physics. A block diagram is shown in Figure 7. The structure reveals a nice implementation of Horner's rule for the evaluation of a polynomial:

$$P_n(x)=c_0+c_1x+c_2x^2+ \,........\, + c_nx^n=$$
$$((\ldots(c_nx+c_{n-1})x+c_{n-2})x+ \,......)x+c_0 \qquad (11)$$

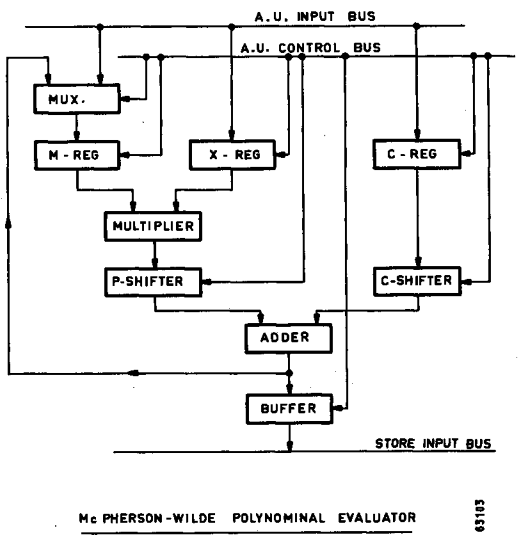


Figure 7 - McPherson-Wilde Polynomial Evaluator

The processor is flexible due to its micro-programmability. It probably will not beat a large computer in speed, but it can be very useful as an extension to a minicomputer without hardware multiplier.

SPECIAL HARDWARE PROCESSORS FOR A SPECTROMETER SET-UP

We will now describe in more detail a number of processors which have been designed (and some built) for track-recognition in a spectrometer set-up using proportional wire chambers. We will describe the processors, investigate why they are faster than programs running on a Control Data 7600, describe how further improvements can be obtained and how the operation of different processors can be co-ordinated to perform the overall task.

Lay-Out of the Experiment

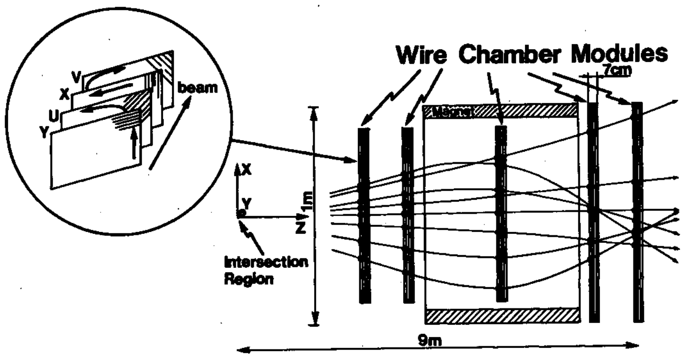


Figure 8 - Schematic of Experimental Lay-out

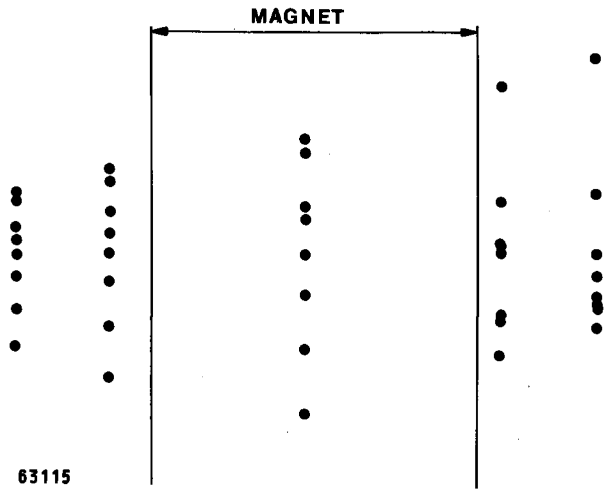


Figure 9 - Same as Figure 8, without tracks

The spectrometer set-up is shown - schematically - in Figure 8. In this set-up one wants to study the secondary particles produced in head-on collisions of two protons. The particles are produced in a relatively small region (∿ 20 x 4 x 2 cm^3), indicated at the left. Note that the vertical scale in Figure 8 is much larger than the horizontal. The secondary particles traverse a number of detectors, where the X and Y coordinates of the traversal point are measured. One of the detectors is placed in a - more or less uniform - magnetic field. Inside the magnetic field the particle trajectories are curved and the amount of curvature is in fact a measure of the momentum. Figure 8 shows a complicated event, where a larger number of positively and negatively charged particles are produced and traverse simultaneously (i.e. within the resolution time ∿ 0.1 μs of the electronics) the set of detectors. The problem we are faced with is: how do we distinguish tracks in this set up? In figure 8 the complete trajectories are drawn in, but the only information we obtain from the experiment are a set of (X,Y)-pairs in each detector. So, it looks in fact as in figure 9 which represents the same event as figure 8! We want to obtain from this, for every particle in the event, a list of five (X,Y)-pairs, representing five points along its trajectory and thus describing this trajectory entirely. This then will allow us to determine the three components of the momentum, make kinematic calculations and analyse the event to extract the physics information of interest.

The problem is slightly more complicated than this, for we do not directly measure (X,Y) coordinate pairs to start off with! The detectors used are multi-wire chambers, consisting of several wire planes. A large number of thin wires are strung parallel to each other in a plane and kept at a positive potential with respect to the outer planes, parallel to the wire-plane and at a distance ∿1 cm. When an ionising particle passes through such a chamber, electron multiplication takes place in the strong electric field around the wire and the wire nearest to the traversal of the particle will produce an electrical signal which can be detected. Since the number of the wire which gave the signal is known, we have measured one coordinate (say X) on the trajectory of the particle. With the help of another chamber, with the wires strung in a direction perpendicular to the first, we can measure the Y coordinate. For a single particle event we thus measure the (X,Y) pair but in a multiparticle event we obtain a set of X-coordinates independently

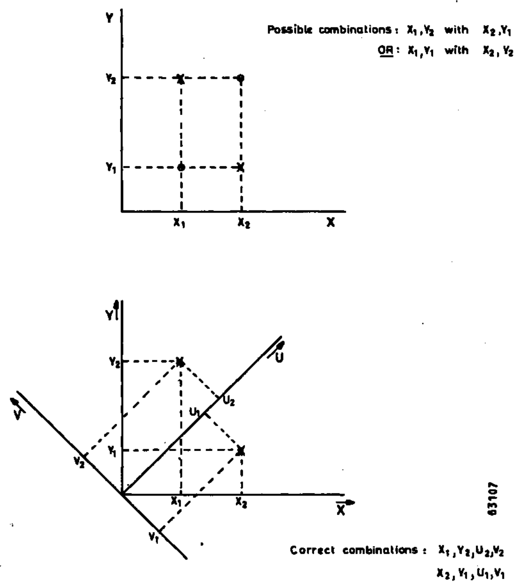

Possible combinations : X_1, Y_2 with X_2, Y_1
OR : X_1, Y_1 with X_2, Y_2
Y
Y_2
Y_1
X_1
X_2
X
U
U_1
U_2
V
V_1
V_2
63107
Correct combinations : X_1, Y_2, U_2, V_2
X_2, Y_1, U_1, V_1

Figure 10 - Ambiguities in Wire Chambers; Use of U,V.

of the Y-coordinates which have to be combined into (X,Y) pairs. This pairing gives rise to ambiguities. To solve the ambiguities that occur in pairing the X's with the Y's (see Figure 10 for the simple case of 2 particles), more wire planes are added and sandwiched together with the X and Y planes in a single module. The added planes have the wires strung under an angle with the X-axis. A typical module is schematized in the inset of figure 8. The wires here are strung at $\pm$ 45^o and they will measure what we will call the U and V coordinates.

To summarize: the electronic read-out system will give us sets of X, Y, U and V coordinates, all independent of each other, for all five detector modules and for a number n particles (a total of 5x4xn coordinates). From this we must find first (X,Y) pairs in each module (a total of 5n (X,Y) pairs). After this point-finding we must sort the points into tracks (to get n tracks).

The Pattern Recognition Process

i) Point-finding. When we apply a simple coordinate transformation from the X-Y-system into the U-V system we see immediately that the four coordinates measured for a particle must satisfy the following relations:

$$U = \tfrac{1}{2}X\sqrt{2}+\tfrac{1}{2}Y\sqrt{2} \qquad (12)$$

$$V = -\tfrac{1}{2}X\sqrt{2}+\tfrac{1}{2}Y\sqrt{2} \qquad (13)$$

In the particular case we are considering the distance between the X and Y wires is 2 mm, but the U and V wires are spaced by 2$\sqrt{2}$ mm. The result of this is that we get rid of the $\sqrt{2}$ in (12) and (13), when we express the coordinates in wire-number, instead of millimeters. The relations (12) and (13) are only satisfied for a particle whose trajectory is perpendicular to the planes. When this is not the case, then due to the finite thickness of a module, (12) and (13) can only be satisfied approximately. So, the four coordinates of a particle must satisfy the following constraints:

$$|X+Y-2U| \leq \varepsilon_1 \qquad (14)$$

$$|-X+Y-2V| \leq \varepsilon_2 \qquad (15)$$

Thus, the problem is to find all combinations of an X, a Y, a U and a V, satisfying (14) and (15) simulteneously. This involves a nest of loops, as in figure 11-i. Matters get somewhat more complicated if we

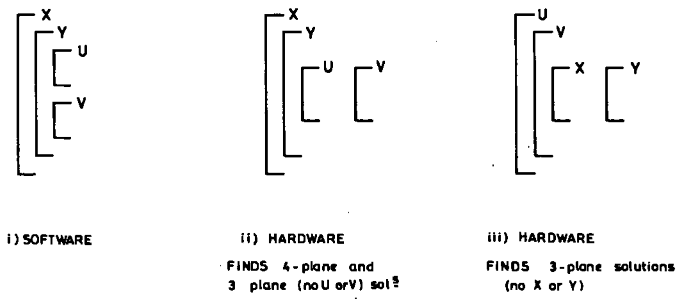

X
Y
U
V
i) SOFTWARE
X
Y
U
V
ii) HARDWARE
FINDS 4-plane and 3 plane (no U or V) sol^s
U
V
X
Y
iii) HARDWARE
FINDS 3-plane solutions (no X or Y)

Figure 11 - Loop Structure for Point-Finding

realize that the efficiency of a detector plane is not exactly 100%. In other words, one can expect to find particles for which, say, the U coordinate is missing, because the particle did not produce an electrical signal on a wire. In that case (15) is the only constraint that can be satisfied. Similarly for a missing V only equation (14) can be satisfied. In order to find also particles for which the X or the Y coordinate is missing, we must use the inverse relationships:

$$|U+V-Y| \leq \varepsilon_3 \qquad (16)$$

$$|U-V-X| \leq \varepsilon_4 \qquad (17)$$

For this purpose we must also perform a search with the loop-structure of figure 11-iii, after executing the loops of figure 11-i.

In real life we must also deal with spurious signals, without any relation with an incident particle. Since all the constraints are to be satisfied within a finite value ε (ε is normally 5 or 6) there is a non-negligable probability of finding spurious solutions. These so called ghosts consist of combinations of X, Y, U and V, which satisfy the constraints by pure chance. Since at the level of a chamber module there is no way of distinguishing a ghost from a true solution, the ghost solutions must be carried through into the track-finding. They increase the number of points participating in the track-recognition process. The number of ghost solutions increases with the square of the total number of signals per wire plane (particle+spurious signals) and the problem can become serious when there are many "noisy" wires.

ii) Track-finding. We now turn our attention to the track-finding process. The method of principal components [43 - 45] offers an elegant method of treating the problem [46 - 47] . Recall figure 9 to realize that the problem is not trivial!

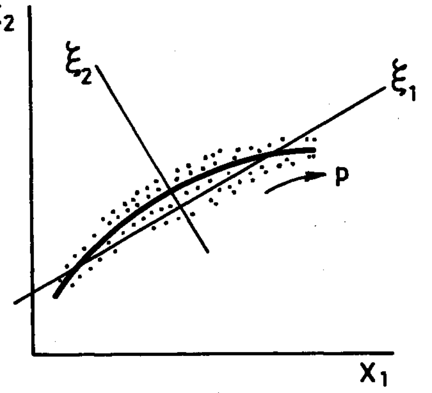


Figure 12 - Principal Components in 2-Dimensions

Instead of presenting immediately a formal mathematical description of the method we will first give an example in two dimensional space. Suppose we measure two quantities x_1 and x_2 to determine the value of a parameter p describing a particle trajectory. We suppose we only want to determine one parameter and therefore the measurement of two variables is redundant. In other words, a constraint equation relates the value of x_1 to the value of x_2. See figure 12 for a representation. The curve represents the constraint, and the parameter p takes on different values along the curve. A measurement of an x_1 and x_2 which does not correspond to a point on the curve (or near to it) must be rejected as non-physical. When the curve is sufficiently close to a straight line (or rather when the probability distribution is well allongated), the test to check if a (x_1, x_2) pair represents a physical situation or not can be simplified by applying a coordinate transformation from the (x_1, x_2) system into the (ξ_1, ξ_2) system (see figure 12). The test criterium then simply becomes: $|\xi_2| < \varepsilon$ where ε is small. The value of ξ_1 is a direct measure of the parameter p.

In the case we are considering of particle trajectories in 3-dimensional space we can apply the same method, but we must increase the dimensions of the space we are working in. In fact, a particle trajectory has 5 degrees of freedom, when a magnetic field is present. Or, to put it differently, a trajectory is entirely determined by 5 parameters: p_x, p_y, p_z (the 3 components of the momentum) and (X,Y) at some plane Z=constant. But we measure 10 values (5 coordinate pairs) along the trajectory in the set-up of figure 8. Thus there must be 5 constraint equations between the measured values $x_1, \ldots, x_{10}$. Each trajectory can be represented by a point $(x_1, \ldots, x_{10})$ in 10-dimensional space. It now turns out that, for a wide range of experimental conditions [45 - 47] the points plotted this way for a large number of tracks, lie close to a 5-dimensional hyperplane. (Compare with figure 12 where the 2 dimensional plot is close to a 1-dimensional "hyperplane"). This, of course, simply means that the constraint equations are linear. Or, in other words, it means that we can define a coordinate transformation

$$\vec{\xi} = \vec{W}.\vec{x} \qquad (18)$$

which transforms from the system $x_1,\ldots.,x_{10}$ into the system $\xi_1,\ldots.,\xi_{10}$ in such a way that the $\xi_1,\ldots.,\xi_5$ axes subtend the hyperplane and that the $\xi_6,\ldots.,\xi_{10}$ axes are perpendicular to it. This then means that for a given trajectory and thus a point $(x_1,\ldots,x_{10})$ in the 10-dimensional space

$$\xi_6^2+\xi_7^2+\xi_8^2+\xi_9^2+\xi_{10}^2$$

is the distance of the point to the hyperplane. This quantity is small for a possible trajectory and can take any value for an arbitrary point in the 10-dimensional space.

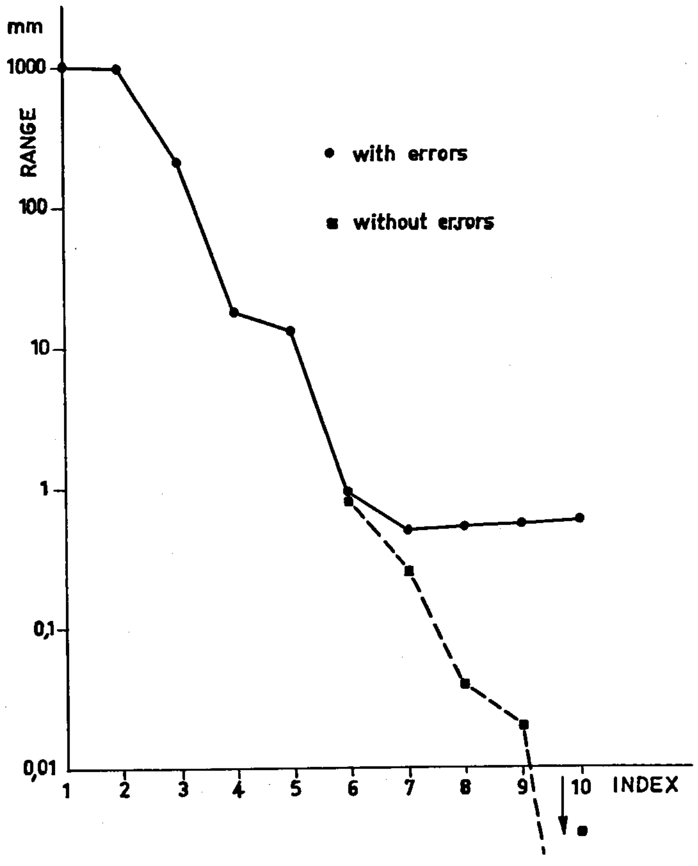


Figure 13 - Range of values for ξ_1 to ξ_{10}

The range of values taken by ξ_1 to ξ_{10} for a set of 1000 tracks is illustrated in figure 13. For each value of the index i is plotted the largest value of $|\xi_i|$ found amongst the sample of 1000 tracks. The dotted line interconnects the points obtained when the original x-coordinates are known without error. This line is a measure of the "flatness" of the hyperplane. When measurement errors are introduced the smaller values of ξ_i, occurring for the larger indices i, are drowned in the errors. The solid line in figure 13 shows this effect.

This now leads to the algorithm to recognize tracks. We first rename our coordinates: (X,Y) in the first chamber module becomes x_1, x_2, in the second module x_3, x_4, etc. We then calculate for every possible combination of one point in the first module, and one in the second, one in the third, etc...

$$\xi_i = \sum_{j=1}^{10} W_{ij} X_j \qquad \text{for } i=6,\ldots,10. \tag{19}$$

When each ξ_i is small:

$$|\xi_i| \leq \varepsilon_i \qquad \text{for } i=6\ldots,10. \tag{20}$$

and/or when

$$\sum_{i=6}^{10} \xi_i^2 \leq \Delta \tag{21}$$

we accept the chosen combination of points as one defining a possible trajectory. If the conditions (20) and (21) are not satisfied we reject the combination.

To apply the method we must of course know the matrix W. This knowledge can be obtained with any precision required from a Monte-Carlo sample of events. W is nothing more than the set of eigen vectors of the covariance matrix of the sample [44], [46] .

A very important thing to know is how well the method does discriminate against wrong combinations. Figure 14 illustrates the quality of the discrimination attained [46] . The peak at the left contains all the correct tracks and the large hill at the right all the wrong combinations. Note that the horizontal scale is linear from 0 to 1 and then becomes logarithmic.

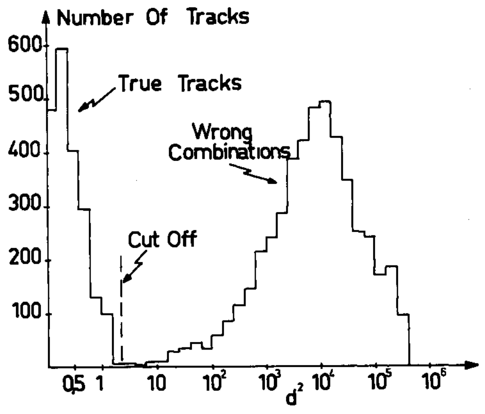


Figure 14 - Separation of Good Tracks

The number of combinations that can be formed with n points in each of the five detectors is n^5 and they should in principle all be tested. Obviously this becomes a big and time consuming task for larger n. The CPU time it takes - in Fortran - on a CDC 7600 is shown in figure 15 (curve marked

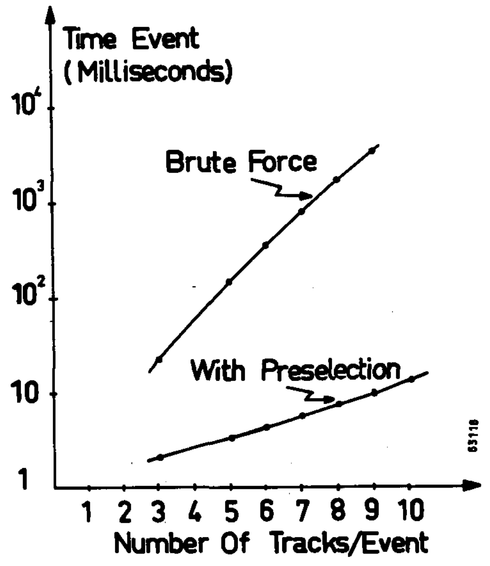


Figure 15 - Timing on CDC 7600

"brute force"). It is therefore essential to reduce the number of combinations to be tested and fortunately this is possible in most practical cases: when the magnetic field is reasonably uniform and the particles have a high momentum, the tracks are close to straight lines in one projection (say Y). A check to see if x_2, x_4, x_6, x_8 and x_{10} lie approximately on a straight line can be done in a time proportional to n^3 (instead of n^5 as with "brute force"). Only for those combinations which pass this straight line test does one then need to calculate (19). Figure 15 shows the gain obtained.

From the point of view of a hardware implementation, this track recognition algorithm is ideal: a straightforward, simple calculation which can be performed in a nested loop structure, followed by simple tests. It should be pointed out, however, that the method has its limitations:

i) the detector planes must be parallel.

ii) the transformation W is only valid for tracks crossing all five detectors.

For other combinations of detectors (the first three, for instance) another transformation must be applied.

Point-Finding Processor

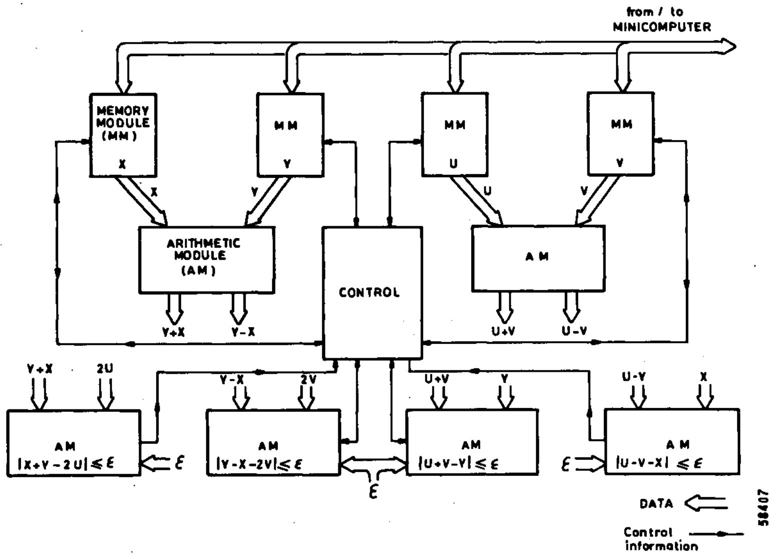


Figure 16 - Block Diagram of Point Finder

The block diagram of a hardware implementation of the point-finding process is shown in Figure 16. This processor works on data from one single 4-plane module at a time. The X, Y, U and V coordinates are stored in independent scratch-pad memories, in the order given by the read-out system, i.e. in increasing order. The scratch-pad memories can store 16 or 32 words of 16 bits. They have their individual address register (AR), word count register (WCR) and number of active coordinates register (ACR). The word count register furnishes in fact a pointer to the last address in memory containing useful data. When a memory is loaded, all three registers are incremented until the last coordinate from the wire plane has been stored and a switch-over is made to the next wire-plane and the next memory. To perform a loop over the data contained in a memory three control signals are enough: set address register to zero (ZAR), increment address register (IAR) and "last address reached" (LA). The last is a signal delivered by the memory module when the contents of AR equal the contents of WCR. It indicates that incrementing of AR must be stopped because the following locations in memory contain rubbish.

Nested loops can be easily implemented in the following way: first all address registers are zeroed. Then AR1 is incremented until the signal LA1 is produced. The presence of LA1 causes the following two actions: ZAR1 and IAR2. This is repeated until LA2 and LA1 are present, causing ZAR1, ZAR2 and IAR3, etc.

Note that the relation (14) and (15) are independent of each other and can be tested simultaneously if the necessary arithmetic elements are available. A sufficient number of adders and comparators have been implemented to make this possible. The loop structure then becomes as in Figure 11-ii. One loops simultaneously over the U and the V coordinates, until a "hit" is found. When this hit is for instance on a U coordinate (which means that (14) is satisfied), one stops incrementing the address of U, but the search over the V coordinates is continued. Two things can happen: either a hit on V is found, or the last address of V is reached without finding a hit. In the first case a 4-plane solution has been found, in the second a 3-plane solution with V missing. The latter has, by the way, a fair chance of being spurious.

The adders and comparators at the right hand side of Figure 16 are a kind of a luxury. They allow us to check relations (16) and (17) simultaneously and thus detect some of the 3-plane solutions with either X or Y missing at an early stage. This proved to be unrealistic and now these elements are only used when a specific search for these 3-plane solutions is made with the loop structure of Figure 11-iii.

Whenever an acceptable solution has been found, the coordinates participating in this solution are tagged by writing a "1" into an unused bit position (11 or 12 bits are sufficient for a coordinate). In addition the solution is written into four scratch pad memories which serve as output buffer. On subsequent passes through the data in a particular memory a tagged coordinate can be treated in two ways:

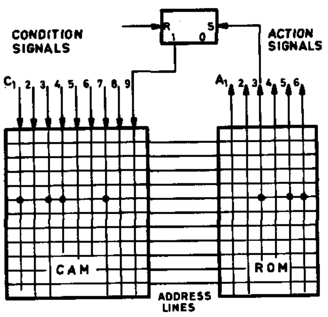
i) The tag can be simply ignored. This will yield all possible solutions, which in the presence of noisy wires can easily amount to 20, 40 or more solutions, with 3 or 4 true particles hidden amongst them.

ii) One can skip rapidly over a tagged coordinate without performing all the arithmetic. A tagged coordinate is thus eliminated and only a limited number of solutions is obtained in which a coordinate appears once and only once. Unfortunately these solutions are not necessarily all correct!

To deal with chambers with noisy wires the following procedure has been adapted: tagged coordinates are ignored. The processor makes four passes through all the data. During pass 1 and 2 only 4-plane hits are accepted. Pass 1 is performed with $\varepsilon=2$ and pass 2 with $\varepsilon=6$. Pass 3, with $\varepsilon=6$ accepts only 3-plane hits where U or V are missing and finally pass 4, also with $\varepsilon=6$, finds the 3-plane hits where X or Y are missing. Since the probability of finding a spurious 3-plane hit is proportional to ε and for a spurious 4-plane hit proportional to ε^2, this strategy reduces the chances of accepting spurious and wrong solutions. (The trouble is not so much the presence of a wrong solution, but the 3 or 4 correct solutions you may miss when tagged coordinates are ignored.)

Another, better, strategy is to suppress pass 1 and to ignore the presence of a tag during pass 2, but not during pass 3 and 4. All possible 4-plane solutions are then accepted and the search for 3-plane hits is performed amongst the residue of unused coordinates.

We realize by now that the box "CONTROL" in Figure 16 has become rather complicated. It has to do all the necessary things to nest the loops, to deviate from normal procedure when a hit on, for instance, U has been found, to organize different passes and different loop structures. By far the optimum way of implementing this control is by the use of a Programmable Logic Array (PLA). A schematic of a PLA is shown in Figure 17. The left hand side, marked CAM, contains AND-gates, one gate per line. The condition signals are the inputs to the AND-gates. In the case of an implementation with AND-gates, each condition C_i should be presented in the straight (C_i) and negated form ($\bar{C}_i$). This allows the use of either one as input to the AND-gate or neither of the two, which represents a "no care" condition. An AND-gate which is satisfied activates an address line which in the part ROM produces the required action signals. The columns in ROM form OR-gates, with a selection from the address



PROGRAMMABLE LOGIC ARRAY
(INSIDE CAM: CHOOSE C_i, $\overline{C_i}$ or "NO CARE")

	1	2	3	4	
CREDIT OK	Y	N	N	N	CONDITIONS
PAY EXPERIENCE OK	–	Y	N	N	
SPECIAL CLEARANCE	–	–	Y	N	
APPROVE ORDER	X	X	X		ACTIONS
REJECT ORDER				X	

"ADDRESSES"

DECISION TABLE

63105

Figure 17 - Programmable Logic Array

lines as inputs. It is important to realize that all the combinations of conditions which have been programmed in the PLA are checked simultaneously and the action signals produced immediately. A stored program computer would have to perform a long series of conditional branches (IF-statements) to do what a PLA does in some 50 ns. Note that a PLA can be programmed to perform certain steps in sequence. It is sufficient that an action signal sets a flip-flop and that the output of this flip-flop is fed-back as a condition signal. In this way the different passes of the point-finding processor have been programmed. While the processor is executing pass 3, the program for pass 3 is enabled while the other three programs are disabled.

Finally, the similarity between a PLA and a decision table [48, 49] should be pointed out (see Figure 17).

Performance of the Point-Finding Processor

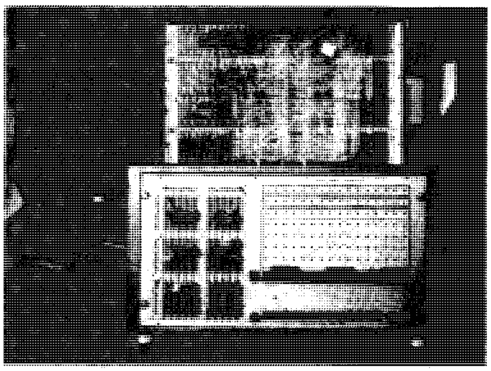


Figure 18 - Photograph of Point-Finder

A photograph of the complete ponint-finding processor as designed and built by A. Fucci, is shown in Figure 18, with all logic circuitry pulled out of the cabinet. The processor is implemented in normal TTL logic and it contains some 540 IC's, distributed over 44 circuit boards. The component cost (including cabinet, switches, etc.) is ∿14.000 SFr., labour for assembly represents ∿5.000 SFr.

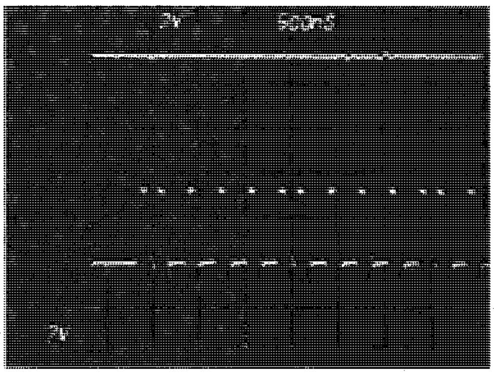


Figure 19 - Long and Short Cycles in Point-Finder

A complete cycle - from the issue of an increment address signal to the next issue of an IAR - is ∿300 ns when a non-tagged coordinate is read and the complete arithmetic is performed. This reduces to ∿130 ns when a skip cycle on a tagged coordinate is performed and it extends to ∿600 ns when a hit is made and a solution must be stored. Figure 19 shows a part of the execution on an extended time scale, beginning at the start of pass 2.

The PLA has been implemented as a diode matrix (see Figure 18) and this implementation accounts mostly for the fact that the cycle time is some 50% longer than intended. The program contained in the PLA has about the complexity of the flow-diagram of Figure 20 (this flow-diagram is not claimed to be correct, it is only included for comparison). The PLA uses 20 input signals and produces 18 output signals; 33 sets of conditions are programmed. The program has been translated back into FORTRAN, resulting in a listing of over 2½ pages, compiling into 397 locations on the CDC 7600.

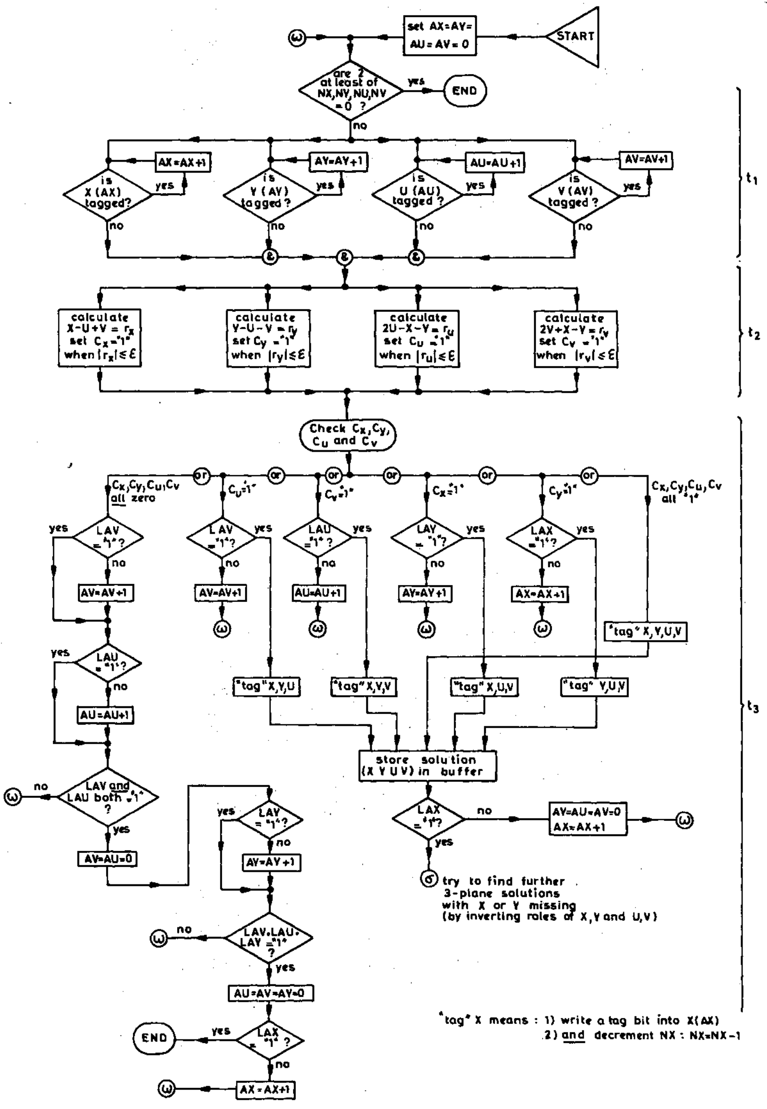

START
set AX=AY=AU=AV=0
are 2 at least of NX,NY,NU,NV =0 ?
yes
END
no
AX=AX+1
is X (AX) tagged?
AY=AY+1
is Y (AY) tagged ?
AU=AU+1
is U (AU) tagged ?
AV=AV+1
is V (AV) tagged ?
&
calculate X-U+V = r_x set C_x="1" when |r_x|≤ε
calculate Y-U-V = r_y set C_y ="1" when |r_y|≤ε
calculate 2U-X-Y = r_u set C_u ="1" when |r_u|≤ε
calculate 2V+X-Y = r_v set C_v ="1" when |r_v|≤ε
t_1
t_2
t_3
Check C_x, C_y, C_u and C_v
or
C_x, C_y, C_u, C_v all zero
C_u="1"
C_v="1"
C_x="1"
C_y="1"
C_x, C_y, C_u, C_v all "1"
LAV ="1"?
LAU ="1"?
LAY ="1"?
LAX ="1"?
AV=AV+1
AU=AU+1
AY=AY+1
AX=AX+1
ω
"tag" X,Y,U,V
"tag" X,Y,U
"tag" X,Y,V
"tag" X,U,V
"tag" Y,U,V
store solution (X Y U V) in buffer
AV=AU=AV=0 AX=AX+1
LAV and LAU both "1" ?
AV=AU=0
LAV.LAU.LAY ="1" ?
AU=AV=AY=0
σ try to find further 3-plane solutions with X or Y missing (by inverting roles of X,Y and U,V)
"tag" X means : 1) write a tag bit into X(AX) 2) and decrement NX : NX=NX-1

Figure 20 - Flow Chart of Point Finding Program

The execution time for the point-finding process is found to be proportional to the third power of the number of signals per plane, as expected. An event with 6 particles is handled in 74 μs (not counting I/O time). This is 25 times faster than the Fortran program on the 7600, compiled with the latest version of the compiler (FTN 4.1 + 69) using full optimization. (For an earlier version of FTN the 7600 performed the algorithm 40 times slower [50 - 52].)

The reasons for this increase in speed can be easily identified:

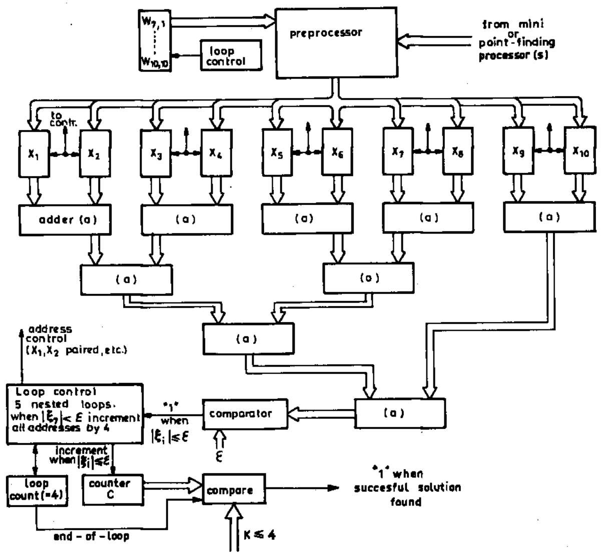
i) parallelism. The use of several scratch-pad memories avoids sequential access to data.

ii) fast access. The memories have an access time of 35 ns, which is increased to $\sim$50 ns because the AR must be incremented. Also the arithmetic is fast: $\sim$35 ns per addition.

iii) duplication. In order to check two relations at a time, the arithmetic has been duplicated. In fact every addition uses a different adder, which saves gating delays at inputs and outputs of adders.

iv) use of PLA. This gives without doubt the most important contribution to the speed-up factor.

Hardware Design for Track Finding



$W_{7,1}$
$W_{10,10}$
loop control
preprocessor
from mini or point-finding processor (s)
to contr.
X_1
X_2
X_3
X_4
X_5
X_6
X_7
X_8
X_9
X_{10}
adder (a)
(a)
address control (X_1,X_2 paired, etc.)
Loop control 5 nested loops. when $|\xi_7| < \varepsilon$ increment all addresses by 4
"1" when $|\xi_i| \leq \varepsilon$
comparator
ε
increment when $|\xi_i| \leq \varepsilon$
loop count (=4)
counter C
compare
"1" when succesful solution found
end-of-loop
$K \leq 4$

Figure 21 - Block Diagram of Track Finder

The algorithm of equations (19), (20) and (21) is easily implemented in a fast hardware processor represented in Figure 21. The trick consists in performing all the multiplications required by (19) beforehand, outside the nested loop. Suppose we want to check only ξ_7, ξ_8, ξ_9 and ξ_{10} and not ξ_6. Suppose also that we limit ourselves to a maximum of 8 points. We will then store in, for instance, the 3rd scratch-pad memory in Figure 21 the following data: at address 0,1, etc.: $W_{7,3}X_3(1)$, $W_{7,3}X_3(2)$, $W_{7,3}X_3(3)$, etc.

At addresses 8, 9, etc.: $W_{8,3}X_3(1)$, $W_{8,3}X_3(2)$, $W_{8,3}X_3(3)$, etc.

At addresses 16, 17, etc.: $W_{9,3}X_3(1)$, $W_{9,3}X_3(2)$, etc.

and finally at addresses 24, 25, etc.: $W_{10,3}X_3(1)$, $W_{10,3}X_3(2)$, etc.

The subscript refers to the i and j in (19), while the index is the ordinal number of the point ($X_3(2)$ is the x coordinate of the second point found in module 2). The point-finding process has established already the linkage between the X and Y coordaintes of each point. We will therefore use all possible combinations of points by performing over the data a series of loops, nested 5 deep (and not 10 deep as Figure 21 might suggest). Moreover, we loop only over the first 8 locations (or less if there are less points) of each memory.

For every combination of points we form in the course of executing the loops, ξ_7 will be calculated by the adder tree. When it turns out that $\xi_7 < \delta$ we have a possible track candidate. ξ_8, ξ_9 and ξ_{10} are then rapidly calculated by increasing <u>all</u> memory addresses by 8. We have still some liberty to choose the selection criteria for accepting a track: require (20) to be satisfied for all i=7,....,10, or require that only 3 relations out of 4 need to be satisfied. These criteria are illustrated in Figure 21. One can also require that (21) be satisfied (using for instance a look-up table to find the square of a small number), or replace the sum of squares by the sum of absolute values. Each coordinate must be multiplied in the beginning by 4 constants. These 4 multiplications can be performed in a micro-second. Thus when the processor is loaded from a minicomputer, no time is lost in the multiplications. The loading of the processor with, for instance, 4 points per chamber module can be performed in 2x4x5= 40 μs. Outputting the final track data can also be done in 40 μs or less. The loops can be executed in a maximum time of $n^5.\tau$ and an average time of $\frac{1}{2}n^5.\tau$. For n=4 particles and a cycle time τ=500 ns we obtain an average time of ½x1024x0,5 μs = 256 μs. This neglects the few microseconds required to check ξ_8, ξ_9 and ξ_{10} once ξ_7 has been accepted. This design can obviously be modified for another number of detectors, another maximum number of points, etc. It suffers however from a few shortcomings:

i) the n^5 dependence of the execution time. For 8 points per module the execution time increases to 8 ms!

ii) the processor is capable of finding only those tracks which have given rise to a point in every chamber module.

This is not necessarily the case for all tracks, either because of inefficiencies or because a track has been bent too much in the magnetic field and it misses the last two detectors. To find these special tracks requires another loop structure (which is not difficult as we have seen), but it also necessitates going back to the beginning and multiplying the coordinates by another matrix W. This is rather awkward in this structure.

More Sophisticated Track-Finding Processor

The remedy to both shortcomings is to perform first a straight-line search on the Y-coordinates alone. This, of course, will give a good pre-selection of track candidates only when the magnetic field is sufficiently uniform so that vertical focussing or defocussing effects are small enough. The search for a straight line can be made in a time proportional to n^3, where n is as always the number of points per detector. Particularly for large events a considerable speed-up can be expected. When a straight line has been found, it must be confirmed that this is indeed a track by performing the 3-dimensional principal components algorithm (19) on the points. Since the selection criterium for finding a straight line will be quite loose, formula (19) will have to be evaluated more often than corresponds to the number of tracks. This drawback is however largely offset by the overall gain in execution time. The special cases of tracks not making the full traversal of all detectors or suffering from inefficiencies can be treated at the level of the straight-line search.

When such a "special" straight line has been found, the combination K of detectors in which it was revealed will then determine immediately the matrix W^K which must be applied in (19) to confirm the existence of a track. The elements of all the possible matrices W can be stored inside the track-confirmation processor. The physicist should however exert some self-discipline in this respect. If in a set up of 8 chambers one wants to detect the tracks which traverse 8, 7, 6, ..., 3 chambers in all possible combinations one needs $1 + \binom{8}{7} + \binom{8}{6} + \binom{8}{5} + \binom{8}{4} + \binom{8}{3} = 219$ different matrices, varying in size from 16x11 down to 6x1, making a total of 8352 coefficients! A careful choice must therefore be made of the limited number of combinations of detectors to be considered.

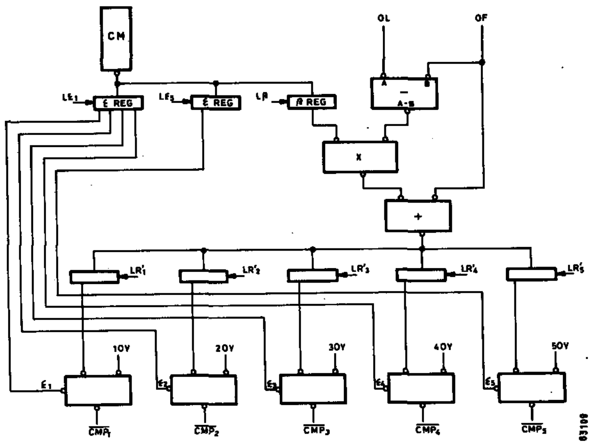


Figure 22 - Diagram of Line Finder

A design for a straight line finder is shown in Figure 22. The algorithm is the following: choose a Y-coordinate in the "first" chamber and one in the "last". Connect the two by a straight line and predict the Y coordinate for the three "intermediate" chambers, using:

$$Y_{pred,i} = Y_{first} + \beta_i \, (Y_{last} - Y_{first}) \qquad (22)$$

The predicted values are then compared with the measured coordinates in the inner chambers. The inner loop over the data for the three intermediate chambers can be performed in parallel. Also the three predictions can be calculated in parallel. In the present design this is done sequentially, to avoid the need for 3 expensive, fast multipliers (2000 SFrs. each). For fast and parallel access the coordinates are stored in scratch-pad memories, one for each detector. So Y_{first} and Y_{last} (OF and OL in Figure 22) are subtracted and multiplied by three β's which are read sequentially from a central memory (CM). The resulting predictions are stored in three of the five registers LR_1 to LR_5.

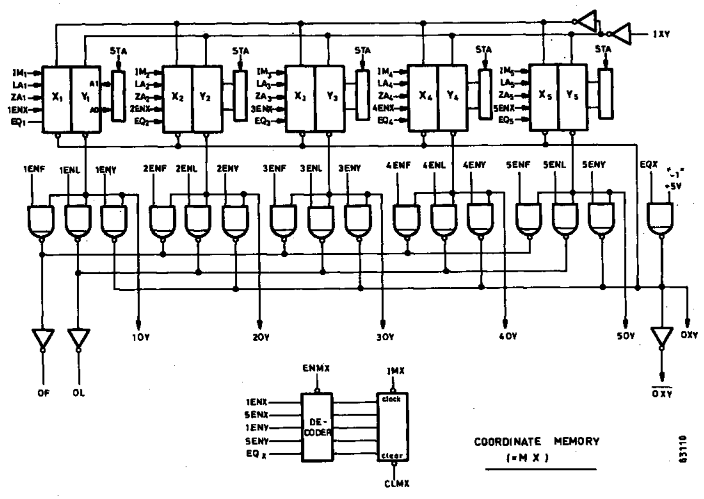


Figure 23 - Selection of First and Last Chamber

The inner loop is then performed by presenting all Y's for the intermediate chambers (2OY, 3OY and 4OY). Figure 22 shows that the choice of the "first" and the "last" chamber is arbitrary. For each choice another set of ε's and β's must be addressed in CM and the control logic must choose the correct combination of LR's and CMP's to be used. (The latter choice can even be avoided by using always 5 values of β, two of which are β=0 and β=1. All 5 LR's and all 5 CMP's will then be used, always.) Figure 23 shows how the choice of first, last and intermediate chambers is made: one of the signals ENF and one of the ENL's are active. Note that the data for the innerloop are available without additional gate delays. Figure 23 shows also the storage for the X-coordinates and the gates necessary to sequentially read out 10 coordinates for track-confirmation.

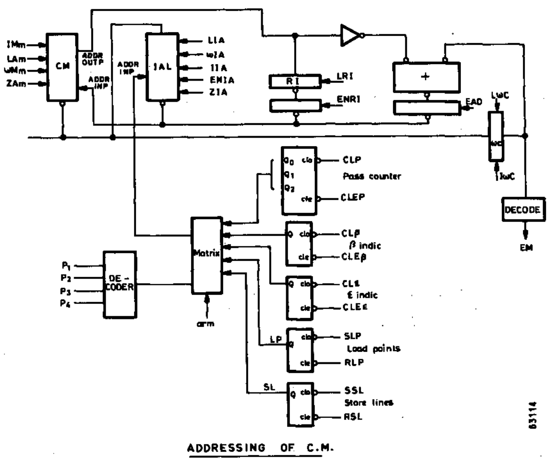


Figure 24 - Indirect Addressing Scheme

In order to deal with the different combinations of ℓ out of m detectors, several passes are performed through the data. A pass counter (see Figure 24) forms the basis for an indirect addressing scheme to retrieve the correct ε's, β's and W_{ii}'s from CM. An undisturbed copy of the starting addresses is kept in CM itself. From here addresses can be transferred into IAL at the beginning of a new pass, if necessary after some manipulation. In IAL the addresses can be freely incremented to perform sequential accesses. The pass counter is not the only variable entering into the addressing scheme. Other factors are the kind of variable that must be retrieved: ε, β or W_{ii}. In addition there is an entry "arm". The whole processor is in fact designed for a two-arm spectrometer. Using the fact that one is treating arm 0 or arm 1 as a parameter allows the access of entirely different constants for the two arms. In other words: the two arms can have different configurations. It is obvious from Figure 22 and 23 that the design of the straight line finder can be easily adapted to any number of detectors. Also a slight modification in the gating in Figure 23 would adapt the processor to find straight lines in X as well as in Y. The line-finder is therefore a processor in itself which could find application in experiments where the principal component approach is not needed.

A preliminary version of this line-finder has been built and it has been tested on Monte-Carlo data. At present it is being tested on real data from an experiment, recorded on magnetic tape. The aim is to compare its performance with the results obtained by software.

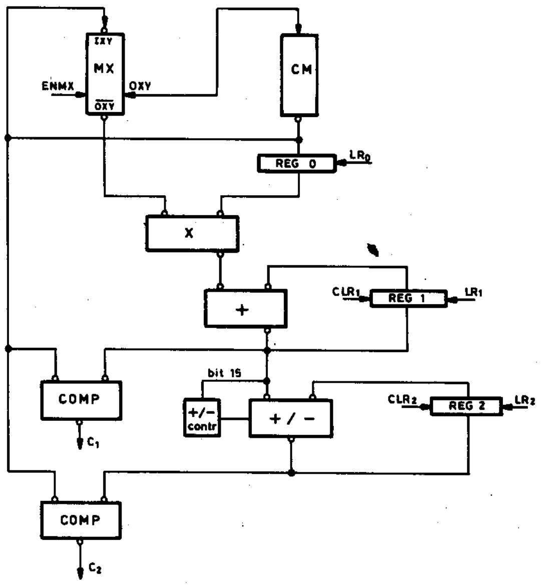


Figure 25 - Diagram of Track Confirmation

The track-confirmation part of the track-finding processor, shown in Figure 25, is simple. The coordinates are obtained sequentially (that is in the order of $X_1, X_2, X_3, \ldots, X_{10}$) from the scratch-pad memories. At the same time the appropriate W_{ii}'s are obtained - also sequentially - from CM. After multiplication the terms are accumulated in REG1. A first comparator compares the ξ_i obtained with an ε_i retrieved from CM. An adder adds or substracts ξ_i depending on its sign from the sum accumulated in REG2. REG2 will thus contain the sum of absolute values $\Sigma|\xi_i|$, which at the end is compared with another ε_{total}.

It is evident that the same processor can be used for track confirmation in an arbitrary number of detectors. It is thus very general.

The results from the track-confirmation are written back into CM, from where they can be transferred to the control computer for recording on magnetic tape. In order to avoid detecting in later passes segments of a track already found in an earlier pass, coordinates which lie on a track are tagged. The tagged coordinates are ignored in later passes where the number of detectors considered is smaller.

The track-finding processor is controlled by PLA's. Instead of one very large PLA controlling everything, the control has been split up. There is one PLA for overall control and a number of others controlling sub-units. The design of the track-finding processor is mainly due to W. Vree.

POSSIBLE IMPROVEMENTS

Use of Content Addressable Memories

Both in the point-finder and line-finder the inner loop consists of the comparison of a predicted with a measured value. This is done by looping over all measured values V_m, subtracting them from the prediction V_p and comparing the absolute value $|V_p - V_m|$ with a small and positive ε. When there are n measured values this takes a time $n\tau$. With the use of a Content Addressable Memory (CAM), this time can be reduced and thus the operation of the processor speeded-up.

A CAM is orthogonal to a random access memory (RAM). To the latter one presents an address and it produces as output the contents of the location addressed. To a CAM a value is presented and, if this value is stored in a location in the CAM, the output produced will be the address of that location. A CAM will therefore indicate if there is an exact match between a predicted value and a measured value, stored together with all other measured values in the CAM. This comparison of the predicted value with <u>all</u> measured values is done in one single memory cycle. An additional facility of a CAM is that certain bit positions can be masked out, so that they do not take part in the comparison. In our case we are not interested really in exact matches, but only in approximate ones, within $\pm\ \varepsilon$. This complicates the use of CAMS.

A first idea that comes to mind is to mask out the least significant bits when a comparison is made. This does not work nicely however. Suppose that the predicted value is 162_8. Masking out the 3 least significant bits will then reveal matches falling in the range 160-167, that is between the asymmetric limits +5 and -2. But things can be even worse: suppose that V_p is 177_8. To find then a match within a range of $\varepsilon = +5$ for instance, a completely different pattern (200_8) must be presented. Sumner has proposed a scheme [53] where two comparisons at most are required, followed by a final precise check using a subtractor and a comparator. Suppose $\varepsilon \leq 4$. At first a comparison is made with V_p, masking out the 3 least significant bits. The 3 least significant bits of V_p are inspected and a second comparison is made either with $V_p - \varepsilon$ when the 3 least significant bits form a number ≥ 4. In both cases the 3 least significant bits are masked out. When a match is found, the final check is made, comparing $|V_p - V_m|$ with ε in full precision. Due to the possibility of finding multiple matches, the control logic becomes quite complicated.

Fucci has proposed another scheme, which is a priori slower, but simpler to implement and where multiple matches present no problem at all. His scheme leads to a new and really splendid design of the point-finder; shown in Figure 26. As in the old design, two programs are executed, one with the outer loops over X and Y, the other with the outer loops over U and V. During execution of the first program any combination of X and Y is handled by the arithmetic to form X+Y and Y-X. These two values are loaded into registers/counters, shown in the top right hand corner. The search for matching U and V values now starts. If a match is found immediately, fine. If not, the register containing Y+X is incremented by one and a new comparison made with the U values. It goes without saying that the search for V procedes in parallel. This is repeated until the value Y+X+ε is reached. Multiple matches within this range will be found in sequence and so present no problem. In reality the register containing Y+X is doubled: one is incremented until Y+X+ε, the other decremented until Y+X-ε.

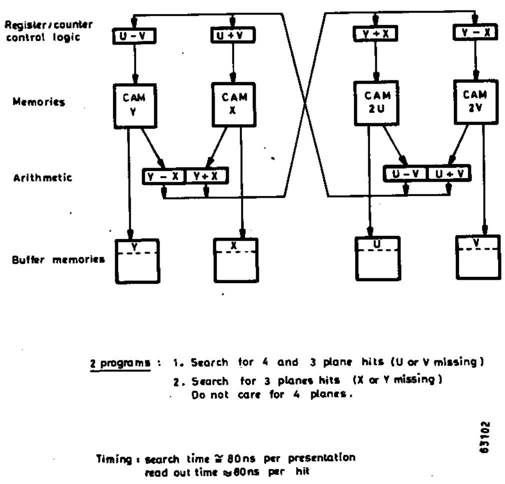

Register/counter control logic
Memories
Arithmetic
Buffer memories
U - V
U + V
Y + X
Y - X
CAM Y
CAM X
CAM 2U
CAM 2V
Y - X
Y+X
U - V
U + V
Y
X
U
V
2 programs : 1. Search for 4 and 3 plane hits (U or V missing)
2. Search for 3 planes hits (X or Y missing)
Do not care for 4 planes.
Timing : search time ≅ 80ns per presentation
read out time ≈ 80ns per hit
63102

Figure 26 - Point Finder Using CAMs

The design provides for overlapping of the search and increment/ decrement operations: while Y+X+ something is presented to the CAM, Y+X-something is decremented and vice-versa. At first sight the gain seems minimal: n comparisons have been replaced by 2ε+1 searches. But the cycle times must be taken into account as well: 320 n ns total time for the inner loop for one, against (2ε+1).80 ns for the other. The method using CAM's is therefore faster whenever $\varepsilon \leq (4n-1)/2$ or $\varepsilon \leq 2$ n.

The second program is executed by looping normally over the U, V memories and performing the search in the X, Y memories. This design still holds another surprise: the two programs can be executed simultaneously! For this it is sufficient to alternate increment/calculate with search, but in both halves of the processor at the same time.

Another use of CAM's should be mentioned, where exact matches are in fact required. This to eliminate a posteriori point or track solutions which have been found already before. If for instance a 3-plane solution is found which is already contained in a 4-plane solution, a search in a CAM containing all solutions would reveal this. The missing coordinate must of course be masked out. It is more economical to work with pointers than with coordinates (less bits).

A sombre note on which to end is that CAM's are expensive and difficult to obtain.

Pipelines

Pipeline structures have been invented to improve the throughput of arithmetic units [55] and of complete CPU (instruction pipeline of MU5, [56]). We are rather interested in pipelining macro-operations to improve throughput. This can often be achieved at the price of increasing the storage capacity inside the processor.

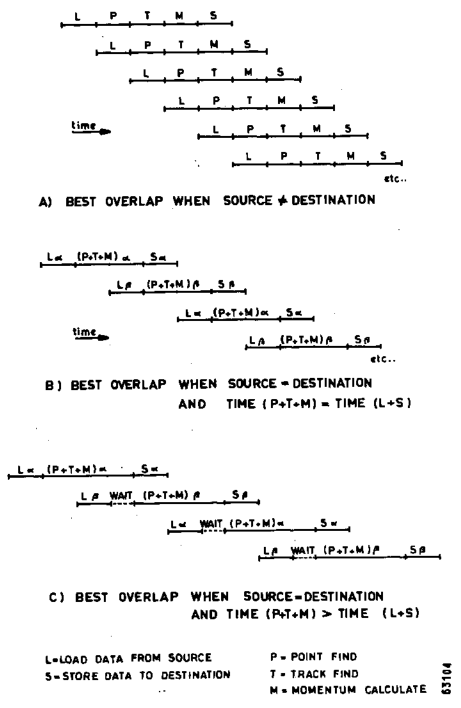

L P T M S
time
etc..
A) BEST OVERLAP WHEN SOURCE ≠ DESTINATION
Lα (P+T+M)α Sα
Lβ (P+T+M)β Sβ
time
etc..
B) BEST OVERLAP WHEN SOURCE = DESTINATION AND TIME (P+T+M) = TIME (L+S)
Lα (P+T+M)α Sα
Lβ WAIT (P+T+M)β Sβ
Lα WAIT (P+T+M)α Sα
C) BEST OVERLAP WHEN SOURCE = DESTINATION AND TIME (P+T+M) > TIME (L+S)
L=LOAD DATA FROM SOURCE
S=STORE DATA TO DESTINATION
P = POINT FIND
T = TRACK FIND
M = MOMENTUM CALCULATE
63104

Figure 27 - Overlap of Processing and I/O

A first example is the overlapping of Input/Output and processing. This requires that the memories for the coordinates be doubled, so that input can proceed into memory I, while processing is done on the contents of memory II. This is illustrated in Figure 27. It is clear from Figure 27 that the effectiveness of pipelining depends on the duration of the different parts. If the time for (P+T+M) would largely exceed (L+S) it might pay off to pipeline point and track-finding. Intermediate storage between the two is then required.

Other examples where something can be gained can be found in the track-finder: when 10 registers are added to store one set of $X_1,\ldots,X_{10}$, track-confirmation and straight line finding can be overlapped. If it pays off depends on the ratio: time to confirm a track/time between finding a line and the next. This ratio is

$$\tau \approx \frac{50\text{x}400 \text{ ns}}{18 \text{ µs}} \approx 1.$$

for 6 points per detector, giving rise to 4 tracks in one single pass. In this particular case, it would pay off, but matters are complicated because $r \propto {}^1/n^2$.

Another example is in the calculation of ξ_i: the multiplication and the addition can be pipelined. But as long as both can be performed sequentially within one CM cycle, there is no point in doing it.

Every possibility for pipelining must be judged on its merits, which in general can only be done when the design is already well advanced.

Cooperation Between Processors

The total pattern recognition process for the spectrometer set-up of Figure 8 can be organized in a number of ways. A single point-finder can handle the five chamber modules one after the other, or five point-finders can be used, operating on the five chambers simultaneously. The choice will depend on a trade-off between the throughput required and the money available. Whatever the choice, track-finding cannot start before all points in all chambers have been found. This asks for some degree of cooperation between the different processors. The cooperation must extend over two fields: overall control and the communication of necessary data. The first can in general be solved by the implementation of an overall control program, using a PLA. The PLA produces start and stop signals, interrupt requests, and it can emit enable and disable signals to other, more local PLA's. It can thus control the processing on the basis of availability of input data, busy or ready states (including the minicomputer), etc.

The communication of data presents other kinds of problems. In our particular case the U and V coordinates were essential for point finding, but they are redundant for track recognition. Should they be discarded when no longer needed? Probably not, because their information content is not negligible and they can contribute to the precision of the final momentum calculation (performed almost certainly on a large scale computer). So they must be stored somewhere. If we continue to use scratch-pad memories for this purpose, we will end up by using really large numbers of them, also at places where the fast access and control facilities are of no use. If in addition we think of overlapping point and track finding these intermediate storage requirements are doubled.

It therefore seems best to communicate data from a processor to the next via a Central Memory, which must be rather big (1K-2K of 16-bit words), but need not have very fast access. A cycle time of 300 ns seems adequate and can be easily obtained with present semiconductor memories. We get a lay-out for the CM more or less as in Figure 28. The general data area can

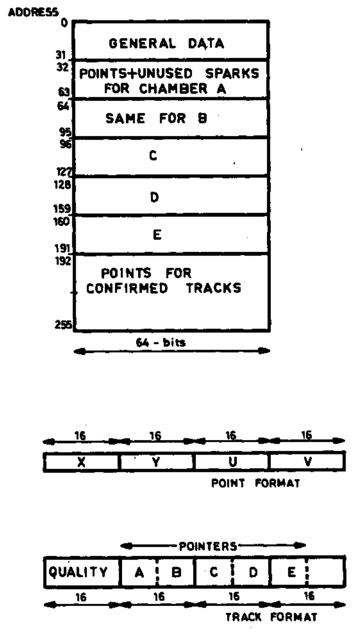


Figure 28 - Memory Layout

have an arbitrary length. It contains general event data, like date, time, scaler read-outs, magnet currents, etc. They are completely useless for the processor, but it is convenient to treat them the same as the other event data. The bulk of the event data consists in fact of "spark" coordinates, which are stored in separate areas. From here they are transferred area after area into the scratch pads of the point-finder. The point-finder writes the results back in the area, overwriting in fact the raw data. Possibly the "unused" data (the spurious signals, not attributable to a point) will be appended. When all points have been found the necessary coordinates are transferred to the straight-line finder and the track confirmation processor, which writes pointers to coordinates back into the last area of CM.

Since access to CM is mostly required for four words at a time, the bandwidth can be improved by making the memory 64 bits wide. The way points and confirmed tracks can be stored is indicated in Figure 28.

USE IN THE EXPERIMENT

Comparison of the input/output times (assuming 1 μs/word DMA transfers without overheads) and inspection of Figure 27 make it clear that maximum throughput can be obtained only when the special processor is placed between the data source (CAMAC) and the data acquisition computer. This however presents some difficulties, not the least important being the necessity that the experimenter has full confidence in the processor. There are others as well: the processor must be loaded with constants and the final values of these constants can only be found by carefully analysing a sample of events. It remains to be investigated how precisely the constants must be known to perform track recognition without any pretention to calculate momenta.

For this reason and for the time being special purpose processors will probably be connected as a peripheral to the minicomputer. Data will therefore pass a number of times through the core memory of this mini. In the experiment for which the point and track finder are being constructed the raw data will in a first stage be written onto a digital video tape. The processors will only intervene during play-back of the video tape. The results from the processors will then be used to select events presenting a desired topology. The selected events will be further analysed, the other discarded (the raw data of the discarded events is however kept on the video tape).

Another point of consideration in the use of processors is the format of the input and more importantly, of the output. There is little or no point in producing the final output in such a cryptic form, that to unpack it a large fraction of the original processing gain is lost. Efficiency of the processor required however that one should store pointers, rather than coordinates at some places. A post-processor in the output data path is the solution. This post-processor would perform with great ease the unpacking and retrieving tasks which are often stumbling blocks in later processing.

In the same way a pre-processor will often be needed. A very good reason for having it is that all our formulae - and therefore the processors - supposed that all coordinates are measured in one unique system. This is not the case and a simple transformation must therefore be applied to every coordinate:

$$X' = K \pm X. \qquad (23)$$

This puts the origin at the right place and it can correct for a read-out which runs the wrong way round!

There may be other reasons why a pre-processor is needed. MEDEA [37] in fact can be considered as a kind of pre-processor.

CONCLUSIONS

An important question remains to be answered: "How much processing on the 7600 do we really gain by having special processors?". At present we can only extrapolate to arrive at a guess. The factor 25 in execution between a processor and Fortran on the 7600 is only applicable to a fraction of the total analysis. Figures from a previous experiment using a similar spectrometer indicate that some 20% of the total time was spent in point and track finding. This increases to ∿30% when no histogramming is done. In this experiment an average of 1½ particle per chamber was found. Assuming the n^3 dependance we extrapolate that for an event with 4-5 points per chamber some 80% of the analysis time will be spent on point and track finding. This fraction of the time will be eliminated by the use of a hardware processor.

Being pessimistic we can thus expect a gain of a factor 2 to 3 in the computer time needed for analysis.

This is valid for the events which are analysed. But much bigger gains can be expected when an event selection can be made on the basis of the topology which is known after track recognition.

The objection which is most often voiced against hardware is its supposed inflexibility. Obviously hardware is "harder" to change than software. (This is not only a disadvantage, it is also "harder" to make a mistake in hardware since there will be a tendency to check carefully before implementing.) We hope to have shown that none of the processors is really dependent on the details of the experimental layout and that they can all be extended or modified to another number of detectors, other relative positions, etc. Also the use of PLA's make it possible to drastically change the logic of a processor in a matter of hours.

All in all, we hope to have shown that special purpose hardware can be designed, and that it can be constructed with todays easily available techniques without having recourse to advanced technologies which are outside the possibilities of a non-specialized laboratory. Moreover we believe that the use of special hardware can save considerable amounts of computer time. But one cannot construct a processor without thinking well beforehand!

Acknowledgements

C. Rubbia first suggested to us the development of special processors for track recognition. We are very grateful for his and P. Zanella's continual interest and encouragement.

The author is grateful to the Institute of Electrical and Electronics Engineers for permission to reproduce Figures 1 (from reference 4), 2 (from reference 17), 4 and 5 (from reference 3). I want to thank my colleagues Adolfo, Brian, Carlo, Joop, Michel, Mike, Paolo, René and Wim for their contributions, helpful discussions and especially for their enthusiastic collaboration. I thank Frank Sumner very warmly for the many things he taught me. Finally I must thank the Organizing Committee for their kind invitation and the secretaries and typists for their hard labour.

References

1. M.R. Schaffner; "Computers formed by the problems rather than problems deformed by the Computers"; Digest of papers 6th annual IEEE Computer Society Int. Conf. (Compcon '72), San Francisco, pp. 259-264.

2. G.D. Bergland; "A guided tour of the Fast Fourier Transform"; IEEE Spectrum, volume 6, pp. 41-52, July '69.

3. H.L. Groginsky and G.A. Works; "A pipeline Fast Fourier Transform"; IEEE Trans. Comput., volume C-19, pp. 1015-1019, 1970.

4. T. Bially; "Signal Processing Applications of the FFT"; Proc. 1971 IEEE Comp. Soc. Int. Conf., Boston, 1971, pp. 7-8.

5. D. Casasent and W. Sterling; "A register transfer module FFT processor for speech analysis"; 1972 Fall Joint Comp. Conf. pp. 709-717.

6. S.H. Unger; "A computer oriented towards spatial problems", Proc. IRE, volume 46, pp. 1744-1750, 1958.

7. B.H. McCormick and R. Marasimhan, "Design of a pattern recognition digital computer with application to the automatic scanning of bubble chamber negatives", in F.J.M. Farley (ed.), Proc. 1962 Conf. Instrumentation for high-energy physics, Geneva, July 1962, pp. 401-406.

8. K. Preston Jr., "Use of the Golay Logic Processor in pattern recognition studies using hexagonal neighbourhood logic", in Proc. Symp. Computers and Automata, New York, 1971, pp. 609-623.

9. B. Kruse, "A parallel picture processing machine", IEEE Trans. Comput, volume C-22, pp. 1075-1087, December '73.

10. C.D. Stamopoulos, "Parallel Algorithm for Joining two points by a straight-line segment", IEEE Trans. Comp., volume C-22, pp. 642-645, June 1974 (contains a short description of CLIP3).

11. G.R. Allen, L.O. Bonrud, J.J. Cosgrove and R.M. Stone, "The design and use of special purpose processors for the machine processing of remotely sensed data", in: Machine Processing of Remotely Sensed Data, (Conf. Proc.), October 16-18, 1973, Purdue, p 1a-25.

12. G.L. Anderson and K. Bartlett, "Hardware allocation of data system resources", Computer Design, volume 13, pp. 89-97, July 1974.

13. U.O. Gagliardi, "Software - related advances in computer hardware", (ACM - Sigarch) Computer Architecture News, volume 3, pp. 11-36, June 1974.

14. G.D. Bergland, "Fast Fourier Transform Hardware implementations - an overview", IEEE Trans. Audio Electroacoust., volume AU-17, pp. 104-108, June 1969.

15. J.W. Cooley and J.W. Tukey, "An algorithm for the machine calculation of complex Fourier Series", Math. Comp. volume 19, pp. 297-301, April 1965.

16. G.D. Bergland, "Fast Fourier Transform hardware implementations - a survey", IEEE Trans. Audio Electroacoust., volume AU-17, pp. 109-119, June 1969.

17. B. Gold, "Efficient computer structures for the FFT algorithm", Proc. 1971 IEEE Int. Comp. Soc. Conf., September 22-24, Boston, pp. 3-4, 1971.

18. R.R. Shively, "A digital processor to generate spectra in real time", IEEE Trans. Comput., volume C-17, pp. 485-491, May 1968.

19. B. Gold, I. Lebow, P. McHugh, C. Rader, "The FDP, a fast programmable signal processor", IEEE Trans. Comput., volume C-20, pp. 33-38, no. 1, January '71.

20. G.D. Bergland and H.W. Hale, "Digital real-time spectral analysis", IEEE Trans. Electr. Comput., volume EC-16, pp. 180-185, April 1967.

21. M.C. Pease III and J. Goldberg, "Feasibility study of a special purpose digital computer for on-line Fourier analysis", Advanced Research Projects Agency. Order 989, May 1967.

22. R.O. Berg and L.L. Kinney, "A digital signal processor", 6th IEEE Comp. Soc. Int. Conf. (Compcon '72), September 12-14, San Francisco, pp. 45-48.

23. R.B. McCullough, "A real-time digital spectrum analyzer", Stanford Electronics Labs. Stanford, Scientific Rep. 23, November 1967.

24. D. Dotti, "A digital instrument for approximate computation of FFT", Alta Frequenza, volume XLII, pp. 214-219, April 1973.

25. T. Lingjaerde and D. Wiskott, "A method of track element search using special purpose digital hardware", CERN/DD/DH/69/7, July 1969.

26. O.P. Fedotov, and D. Wiskott, "LER, a device for line element recognition from digitized points output from flying spot scanners", CERN, DD/71/2, January 1971.

27. H. Billing, A. Rüdiger and R. Schilling, "BRUSH - an economical special purpose computer for on-line HPD data-processing", Proc. Int. Conf. on Advanced data processing for bubble and spark chambers, Argonne, Oct. 28-30, 1968, pp. 48-80.

28. H. Billing, A. Rüdiger, R. Schilling. L. Schnupp, R. Kaufhold and F. Gandini, "The BRUSH system for automatic processing of bubble chamber pictures", Int. Conf. on data handling systems in high-energy physics, Cambridge, March 23-25, 1970, pp. 689-706.

29. A. Rüdiger and R. Schilling, "BRUSH picture processing with rapid cycling systems", Proc. Seminar on track analysis for rapid cycling bubble chambers, Rutherford Lab. 14 February 1973, RHEL/R 271, pp. 120-132.

30. M.A. Thompson, "SATR - an automatic scanning and measuring machine", Proc. 1966 Int. Conf. Instrumentation for High-Energy Physics, Stanford, September 9-10, 1968, pp. 312-326.

31. M.A. Thompson, "The hardware of SATR", Proc. Int. Conf. on Advanced Data Proc. for Bubble and Spark Chambers, Argonne, October 28-30, 1968, pp. 81-101.

32. B. Equer, C. Güignard, G. Reboul and A. Volte, "Coccinelle, a device for automatic pattern recognition and event measurement on bubble chamber photographs", Proc. Int. Conf. on Advanced Data Processing for Bubble and Spark Chambers, Argonne, October 28-30, 1968, pp. 6-20.

33. B. Equer, G. Fontaine, and G. Reboul, "The use of Coccinelle, a CRT digitizer to scan and measure big bubble chamber pictures", Int. Conf. on Data Handling Systems in High-Energy Physics, Cambridge, March 23-25, 1970, pp. 371-380.

34. P. Antoine, J. Cech, C. Guignard, M. Guiollot, M. Gutmann, C. Ouannès, J. Passeneau, M. Prinzie, and M.J. Robert, "PANGLOSS - a CRT device for bubble chamber pictures", Int. Conf. on Data Handling Systems in High-Energy Physics, Cambridge, March 23-25, 1970, pp. 401-424.

35. L.O. Hertzberger and W. Vree, private communication.

36. T. Lingjaerde, Ljusljin and D. Marland, private communication.

37. H.J. Stuckenberg, "MEDEA, ein Spezialrechner zur Vorbearbeitung und Formatierung von Proportionalkammer - Daten", DESY 73/54, November 1973.

38. A.A. Derevshikov, Z. Guzik, Y.A. Matulenko, S.B. Nurushev, E.V. Smirnov, V.L. Solovyanov, "Digital device for high-energy experiments", Serpukhov, 1972, pp. 8, IHEP 72-4 (in Russian).

39. Th. A. Nunamaker and J.T. Solomon, "Pattern recognition processor and its application to straight line reconstruction of spark chamber data", Nucl. Instr. and Methods, volume 107, pp. 15-20, 1973.

40. J.T. Solomon, "Pattern recognition processor", Proc. Seminar on Track Analysis for Rapid Cycling Bubble Chambers, Rutherford Lab., 14 February, 1973, REHL/R 271, pp. 134-153.

41. M. Schwartz, R. Flexer, L. Birkwood, R. Spitzer and S. Hertzbach, private communication.

42. G. McPherson and P. Wilde, " A polynomial evaluator", Proc. Seminar on Track Analysis for Rapid Cycling Bubble Chambers, Rutherford Lab., 14 February, 1973, RHEL/R 271, pp. 176-184.

43. H.C. Andrews, "Introduction to mathematical techniques in pattern recognition", Ch. 2, p. 15, Wiley Interscience, New York, 1972.

44. J.H. Friedman, "Data Analysis and Presentation", these Proceedings, pp. 271

45. H. Wind, "Function parametrization", Proc. 1972 CERN Computing and Data Processing School, CERN 72-21, pp. 53-106.

46. H. Grote, M. Hansroul, J.C. Lasalle, P. Zanella, "Identification of digitized particle trajectories", Proc. Int. Computing Symp. 1973, September 6-7, Davos, pp. 413-421.

47. M. Hansroul, D. Townsend, P. Zanella, "The application of multi-dimensional analysis techniques to the processing of event data from large spectrometers", presented at: Meeting on Programming and Mathematical Methods for

Solving Physical Problems, Dubna, October 30 - November 3, 1973. Also: CERN/DD/73/31 October 1973.

48. U.W. Pooch, "Translation of decision tables", Computing Survey, volume 6, pp. 125-151, June 1974.

49. T.R. Gildersleeve, "Decision Tables and Their Practical Application in Data Processing", Prentice Hall, Englewood Cliffs, N.J. 1970.

50. M. Hansroul, C. Verkerk, "Point and track-finding processors for multi-wire chambers", Proc. Seminar on Track Analysis for Fast Cycling Bubble Chambers, Rutherford Lab.,, 14 February, 1973, RHEL/R 271, pp. 155-173.

51. M. Hansroul, C. Verkerk and P. Zanella, "Hardware processors for pattern recognition tasks in wire chamber data", Int. Conf. on Instrumentation for High-Energy Physics, Frascatti, 8-12 May, 1973, pp. 497-499.

52. A. Fucci, M.F. Letheren, F.H. Sumner, C. Verkerk and W.G. Vree, "Design of Hardware processors for use in high-energy physics", to be presented at 1974 Conference on Computer Systems Technology, London 29 October - 2 November 1974.

53. F.H. Sumner, private communication.

54. A. Fucci, private communication.

55. J.E. Thornton, "Design of a Computer; the Control Data 6600", Scott, Foresman and Cy, Glenview, Illinois, 1970.

56. F.H. Sumner, "The architecture of the MU5", these proceedings, pp. 65

57. C.M. Fisher, R.A. Lawes and B.W. Powell, "A vidicon trigger and data processing system for a rapid cycling bubble chamber", Proc. Seminar on Track Analysis for Rapid Cycling Bubble Chambers, 14 February, 1973, Rutherford Laboratory, RHEL/R 271, pp. 185-199.

58. W. Jank, "ERASME, a bubble chamber film measuring system", these Proceedings, pp. 367