POINT AND TRACK-FINDING PROCESSORS

FOR MULTIWIRE CHAMBERS

Presented at a seminar on "Track Analysis with Rapid Cycling Bubble Chambers", held at the Rutherford Laboratory, on 14th February, 1973.
POINT AND TRACK-FINDING PROCESSORS FOR MULTIWIRE CHAMBERS

M. Hansroul and C. Verkerk

CERN - Geneva

The hardware processors described below are designed to be used in conjunction with multi-wire chambers. They have the characteristic of being based on computational methods in contrast to analogue procedures. In a sense, they are hardware implementations of computer programs. But, being specially designed for their purpose, they are free of the restrictions imposed by the architecture of the computer on which the equivalent program is to run. The parallelism inherent in the algorithms can thus be fully exploited. Combined with the use of fast access scratch-pad memories and the non-sequential nature of the control program, the parallelism accounts for the fact that these processors are expected to execute 2-3 orders of magnitude faster than the equivalent Fortran programs on a CDC 7600 or 6600. As a consequence, methods which are simple and straightforward, but which are impractical because they require an exorbitant amount of computer time can on the contrary be very attractive for hardware implementation. To find particle trajectories, for instance, a systematic search through all possible combinations of co-ordinates will take a prohibitively long time on a computer when the number of tracks and the number of points per track are larger than 3 or 4. But hardware implementation is perfectly feasible and very fast, as will be shown later.

1. Point-finding processor

The simplest arrangement of wire chambers allowing to resolve ambiguities due to multiple particles consists of 3 planes with wires running in different directions. For a particle crossing the planes more or less perpendicularly the following relation between the three measured co-ordinates (see fig.1) holds:

\[ |C^p - C^q + C^r - q| \leq \epsilon \]  \hspace{1cm} (1)

When, as in fig. 1, the wire directions are 0°, 60° and -60°, \( C^p = C^q = C^r = 1 \). One can always make \( \alpha = 0 \) by suitable choosing the origins of \( P, Q \) or \( R \).

A computer program to find the points corresponding to the particle traversals would try different combinations of \( P, Q \) and \( R \) and check if relation (1) is satisfied. In Fortran it would look like:

...
DO 10 IP = 1, NP
DO 10 IQ = 1, NQ
DO 10 IR = 1, NR
IF (ABS (P(IP)+Q(IQ)+R(IR)) GT EPSIL) GO TO 10

C FOUND stores the result somewhere.
CALL FOUND
10 CONTINUE

The program essentially consists of 3 nested do-loops, with a very simple calculation and test in the inner loop. The algorithm can be readily implemented in hardware, as the block diagram if fig.2 shows. Three scratch-pad memories contain the P, Q and R co-ordinates (already transformed by the pre-processor, if needed). Each memory is incorporated in a module which contains the necessary controls, like address register and word count. Not more than 3 signals are needed to control a loop over the contents of such a memory module: ZAP (reset to zero the address of P), IAP (increment address of P) and LAP (last address containing a valid P has been reached). The arithmetic module contains two adders and a magnitude comparator and delivers a control signal C = "1" when (1) is satisfied. Otherwise C = "0". The control signals are sampled at the end of each clock period and the box "CONTROL" decides on the action to take. It can be readily verified that the actions enumerated if fig. 2 will cause the execution of the 3 nested do-loops as long as C = "0" (initial condition is AP=AQ=AR=0, of course). When a valid solution has been found, the co-ordinates contributing to it are available at the outputs of the memory modules and can immediately be stored elsewhere. The loops are then taken up again by resetting the addresses of Q and R and incrementing the address of P.

If desired, the co-ordinates which have contributed to a solution can be "removed" from the memories by attaching a tag bit to them. The hardware is designed so that tagged co-ordinates may be rapidly skipped. A skip cycle will take approximately 75ns for an implementation in TTL logic. A full calculate cycle will be around 175ns. To find N particles, the inner loop must be executed \( \frac{1}{3} N^3 \) times (on average). The processing time for 16 particles and an average cycle of 125ns will then be 256\( \mu \)s and for 8 particles 32\( \mu \)s.

.../
This processor will stop when one of the memories becomes empty. So, in principle, particles giving a signal on only two planes out of three cannot be detected.

To increase the efficiency wire chambers with four planes can be used. Accepting also particles which gave a bit on 3 planes only, brings the overall efficiency of such a wire chamber module very close to 100%, so that no particle escapes undetected. With U and V wires at ±45° and a wire spacing \( \sqrt{2} \) times wider than for the X, Y planes, the following two relations must be satisfied for a particle crossing the planes perpendicularly:

\[
\begin{align*}
2U &= Y + X \\
2V &= Y - X
\end{align*}
\]

(2) and (3) can be checked independently and although there are 4 do-loops, they are nested only 3 deep:

\[
\begin{align*}
\text{DO....IX=} \\
\text{DO....IY=} \\
&\quad \text{DO....IU=} \\
&\quad \quad \text{continue} \\
&\quad \text{DO....IV=} \\
&\quad \quad \text{continue}
\end{align*}
\]

The general structure of a hardware processor to implement this algorithm is an extension of the 3 plane processor (see fig. 3). Note that (2) and (3) are checked simultaneously as well as the two reverse relations. The latter do not add anything for a particle which gave a hit on all 4 planes, but they are very useful for finding as early as possible the 3 plane hits. The principle of operation is the same as for the previous processor, but "CONTROL" is more complicated. A kind of a flow-diagram of the principal part of its action is shown in figure 4. It should be noted that the actions grouped under \( t_1 \) take place simultaneously and so for \( t_2 \) and \( t_3 \).

For the prototype - about to be ready for testing at the time of writing - "CONTROL" is implemented as an alterable diode matrix. This can be considered as a combination of a Content Addressable Memory and a Read Only Memory. For every set of conditions the CAM furnishes an address to the ROM. The word
read from the ROM specifies the actions to take. The diode matrix thus specifies the "program" of the processor, but there is no fixed sequence in this "program".

The cycle time of this point-finding processor is the same as for the previous one, and 16 particles can be found in 256µs. It is instructive to compare this figure with the time needed to load the co-ordinates from a mini-computer and to read the results: 128µs at the best. For 8 particles: processing time = 32µs, I/O time = 64µs.

A Fortran program, performing the same algorithm runs = 10³ times slower on a CDC 6600.

2. Global method for track-finding

Wind³⁴ has described a method for calculating relevant parameters of particle trajectories in the presence of a magnetic field (not necessarily homogenous or extending over the full length of the trajectory). The various matrices and polynomials required to calculate the momentum (or any other parameter) are obtained from a Monte-Carlo sample of tracks. We will not describe how these matrices etc. are found, but the application of the method will be briefly outlined.

Suppose five space points (x, y, z) of a trajectory have been measured. Of the 10 variables X₁,...,X₅, Y₁,...,Y₅ five are sufficient to describe the trajectory. Thus the momentum can be expressed as

\[ p = F(\xi_1, ..., \xi_5). \] (4)

The \( \xi \)'s are obtained by a linear transformation from \( \bar{x}(X_1, ..., X_5, Y_1, ..., Y_5) \):

\[ \xi = W \cdot x. \] This transformation W has been obtained from the Monte-Carlo tracks by the method of principal components. This consists in diagonalizing the correlation matrix \( A_{ij} = \sum_{n=1}^{NT} x_i(n) \cdot x_j(n) \) (where \( n \) is the track number; NT the total number of tracks) and finding its eigenvectors. Each column of \( W \) is then an eigenvector of \( A \). By ordering the non-negative eigenvalues \( \lambda_i \) and their corresponding eigenvectors \( W_i \) in descending order, one compresses the original information into \( \xi_1 \) to \( \xi_5 \), whereas \( \xi_6 \) to \( \xi_{10} \) are redundant variables which are small and represent the available constraints.
Therefore, the fact that $\xi_6 \ldots \xi_{10}$ are small can be used as a criterion to decide if $X$ lies on a possible trajectory or not. In practice "small" means: of the same order as the measurement error on the original co-ordinates.

We have used Wind's method for a set-up consisting of five wire-chambers: two in front, one inside and two behind a magnet (fig. 5). We generated 1004 Monte Carlo tracks with momenta from 3 to 32 GeV/c and filling uniformly the solid angle covered. The ranges of values the different $\xi$'s assume are shown in fig. 6. The measurement error introduced was $\pm 0.3$ mm. We then grouped the trajectories into events, containing from 2 to 8 particles. After sorting each co-ordinate in ascending order, (as they would have come out of a wire chamber read-out system) we calculated:

$$\xi_6, \ldots, \xi_{10} \text{ from } (X_1Y_1)_i, (X_2Y_2)_j, (X_3Y_3)_k, (X_4Y_4)_l, \text{ and } (X_5Y_5)_m$$

for all the combinations of $i$, $j$, $k$, $l$, $m$ (each of $i$, $j$, $k$, $l$, $m$ can take the values 1, ..., N with N the number of particles in the event). A solution was retained whenever four out of the five variables $\xi_6 \ldots \xi_{10}$ were smaller than two to four times the measurement error. All 1004 tracks were correctly recognized in this way. In addition we found four more tracks. (In two events two tracks were very close to each other; in fact in some points too close to be separable in a real chamber. In both events these two tracks gave rise to four solutions. Using other selection criteria the four additional tracks would probably have been eliminated.) As an example, the behaviour of one of the $\xi$'s for accepted and rejected combinations is shown in fig. 7. It took 200 seconds of 7600 CPU-time to obtain these results for 1000 tracks. Considering that the time per event is proportional to $N^5$ ($N$ is number of tracks) this means that a 4-prong event requires more than 100 ms of 7600 CPU-time. The program was written in Fortran. These results were obtained on positive tracks only. A further study is in course, where both positive and negative tracks are present in each 10-prong event. The large number of tracks per event necessitated that measures be taken to reduce computer time. The pre-selection introduced need not be implemented in hardware, however.

The results of this study so far confirm entirely the conclusions reached before. For instance, in 200 10-prong events, 199 events have been recognized correctly. One event gave rise to two extra tracks. The reason (and the remedy!) is the same as above. See track 7 and 8 in fig. 9.

.../
We must remark that the matrix W has been found supposing that five points are known on the trajectory. When a point is missing, another matrix must be used. The method therefore requires detectors with a very good efficiency.

3. Hardware Processor for track recognition

A hardware processor implementing the method outlined above is shown in figure 8. The implementation is for five points on a trajectory. It checks only four out of the five possible \( \xi \)'s, but the memories can be easily extended to 32 words and the addressing scheme modified so that all of \( \xi_6 \) to \( \xi_{10} \) can be checked for up to six particles. The preprocessor in fig. 8 performs once and for all the multiplications \( W_{ij} X_j(k) \) required for the transformation \( \xi_1 = \sum W_{ij} X_j \). \( k \) is the index of the points; we assume that \( k=1, \ldots, 4 \). The preprocessor is active during the filling of the memories and the multiplication can be performed inside the time (1μs) required to transfer a word from a minicomputer. So it takes 160μs to load the track-finding processor. (When loaded from a point-finding processor this reduces to less than 80μs. Moreover operations could then be overlapped.) The memories are loaded in a particular way. The first one, for instance, contains:

\[
W_{7,1} X_1(1), W_{7,1} X_1(2), W_{7,1} X_1(3), W_{7,1} X_1(4), W_{8,1} X_1(1), \ldots, W_{10,1} X_1(4)
\]

in that order. Once loaded, the processor performs do-loops, nested five deep. The addresses of an odd and even memory are stepped up together, since they contain the \((X,Y)\) pair of a point, supposed to have been correlated before. The loops extend over not more than the first four locations of each memory. The tree of adders calculates for every combination \( k1, \ldots, k10 \):

\[
\xi_7 = W_{7,1} X_1(k_1) + \ldots + W_{7,10} X_{10}(k_{10}).
\]

The result \( \xi_7 \) is checked against a value \( \epsilon \). In case \( |\xi_7| \leq \epsilon \) the combination of points is accepted as a candidate for a trajectory. All memory addresses are now incremented by four and the output of the adder tree will become \( \xi_8 \), which will be checked in turn. The same for \( \xi_9, \xi_{10} \). A trajectory will be accepted when all four \( \xi \)'s are smaller than \( \epsilon \), or three out of four, or when the sum of squares is less than \( \Delta \) or any other suitable criterion.
The processing time will be on average $T = \frac{1}{2} N^5 \tau$. For $N=4$ and a cycle time $\tau = 250\text{ns}$, $T = 128\mu\text{s}$. A gain in speed of a factor $=500$ over a Fortran program on the 7600 is obtained! The coefficients $W$ are $\leq 1$ and each $\xi$ must be compared with an $\epsilon$ which is $=1$. So it turns out that all calculations can be done in 16-bit precision. This is a very important fact, of course.

4. Conclusion

A comparison of the three processors described shows that they are very similar. They turn out to contain a limited number of building-blocks, like memory and arithmetic modules. Without doubt a coherent system of building-blocks can be designed for easily constructing these, and other, processors. With some design automation the physicist could have a powerful means of adapting processors to his needs and to set up systems for very fast data analysis. With the addition of multinomial evaluators to the catalogue of modules, events could be partially analyzed by hardware. It is not inconceivable that the decision to retain an event could in future be made in real time on the basis of this partial analysis. It could well be that hardware processors are at the same point where the on-line computer was a decade ago: at the bottom of a steep upward slope.

5. Acknowledgements

C. Rubbia is the originator of the basic idea of using arithmetic methods. He, and P. Zanella, have supported our design efforts with their enthusiasm. We are greatly indebted to both of them. We thank H. Wind and D. Townsend for enlightening discussions of the track recognition method. We are grateful to A. Fucci and C. Mazzari, who competently transformed the block diagrams into reality.
FIGURE CAPTIONS

Fig. 1. Definition of co-ordinate system in a 3-plane wire chamber

Fig. 2. Block diagram of a hardware processor for point-finding in a 3-plane wire chamber.

Fig. 3. Hardware processor for a 4-plane wire chamber.

Fig. 4. Diagram showing the principal part of the logic of the processor of Fig. 3.

Fig. 5. Lay-out of an experiment used for the Monte-Carlo calculations.

Fig. 6. Range of values taken by $\xi_1$ to $\xi_{10}$ when the measurement error is supposed to be 0.3 mm.

Fig. 7. Examples of distributions of $\xi$.

Fig. 8. Hardware processor for recognizing tracks in the set-up of Fig. 5.

Fig. 9. A 10-prong event in which two extra tracks were found.
REFERENCES

1. J. Solomon and Th. A. Nunamaker; Pattern recognition processor and its application to straight line reconstruction of spark chamber data; preprint, submitted to Nucl. Instr. and Methods, private communication.

2. E. Iarocci and P. Walorschek; Hardware Logic for the selection and analysis of events observed in Charpak Chamber and counter experiments, DESY 72/13, March 1972.


Fig. 1
Coordinate system for a 3-plane chamber
Control lines:
from/to memory P:
LAP = "1" when last address containing data is reached
ZAP : set address of P to zero
IAP : increment address of P
similar for Q & R

arithmetic + comparator
|P+Q+R| ≤ ε?

when C = '0': IAR
when C.LAR = '1': ZAR, IAR
when C.LAR.LAQ = '1': ZAR, ZAQ, IAP
when C.LAR.LAQ.LAP = '1': end
when C = '1': valid solution, output and tag.

Fig. 2
Fig. 3
Control information
Fig. 4

*tag* X means:
1) write a tag bit into X(AK)
2) and decrement NX : NX = NX - 1
Fig. 6

Range of values taken by $\xi_1$ to $\xi_{10}$

- with measurement error
- without
Fig. 7 Distributions of $\xi_7$
Fig. 9  10-prong event in which 2 extra tracks were found