Massively Parallel Associative String Processor (ASP) for High Energy Physics

G. Vesztergombi
Central Research Institute for Physics
KFKI-RMKI, Budapest, Hungary

Abstract

Tracking and clustering algorithms for massively parallel string processor are presented and their implementation on the ASTRA machine is demonstrated.

1. INTRODUCTION

Parallel computing is one of the "standard item" on the agenda of the Cern Schools of Computing. See, for example, the excellent lectures in previous proceedings [1],[2],[3]. In this lecture I should like to concentrate the attention to a special type of parallel systems and demonstrate the basic ideas in a specific realization, the ASTRA machine. This machine was presented in this School on a facultative tutorial session by F. Rohrbach for the interested students and after the School it is continuously available at CERN for registered local and remote users [4].

In order to better understand the spirit of this lecture I should like to emphasize that I am not a computer specialist, but a physicist who was fascinated by the opportunity to solve experimental physics problems by mapping them onto a new unusual computer architecture.

1.1 Why ever faster computers?

Paolo Zanella asked this question in his '92 lecture. His answer was: "There are many problems which seem to have **unlimited** appetite for Mips and Mflops". I find today the only difference is that one hears more about Gips and Teraflops. This appetite is well illustrated by the simple observation: the "small" increase of the problem size demands "much faster" rise in the amount of computation. For instance, the calculation of the product of two matrices of size n requires n^3 operations. That is, the problem **doubling** requires an **eight-fold** increase in computation time.

In High Energy Physics (HEP) the ever increasing energy, luminosity and event complexity are the driving forces for such unsatisfiable appetite:

a) TRIGGER problem: In LHC pp collisions one should select **by intelligent trigger** only a few events from the milliards of occurring interactions;
b) DATA VOLUME problem: In LHC heavy ion collisions one expects to record events with 10 - 20 thousands of particles emerging from a **single** interaction, producing very fast 1000 TeraBytes of information even at rather modest luminosities.

c) SIMULATION problem: For the correct data evaluation and background calculation one uses simulated events. The simulation of the complex detectors with

millions of active electronic channels at the necessary level of accuracy requires even more computing power than the processing the experimental data itself.

In this situation it is no wonder why are the particle physicists so keen about the increase of computing capacity.

1.2 Classification

One way to increase computing capacity: GO PARALLEL! If you wish huge increase: GO **MASSIVELY** PARALLEL! Using 1000 processing elements of capacity which is similar to the microprocessors in modern RISC workstations really represents a quantum-jump in performance, but here I should like to turn the attention toward an other possibility where one goes even further applying million(s) of processors, of course, with very much reduced individual capacity per processor. In order to put this idea in a more general context Fig.1 shows a slightly modified version of Flynn's classification diagram taken from ref. [1]. Corresponding to the single versus multiple and instruction versus data stream choices, according to Flynn's classification one can have four basic multi-processor architectures: SISD (= Von Neumann machine), SIMD, MISD and MIMD. In the next we shall deal mainly with SIMD-type systems, where all processors execute the **same instruction** synchronously on the data stored **locally**. In the original figure the SIMD class contained only two main sub-classes: Pipelined + Vector and Parallel Array Computers. It seems to be reasonable to distinguish the simple string case from the more general array case, because combining two features: cheap (1-bit) processors **AND** extremely simple (1-dimensional) string-type inter-processor communication one can really think about parallelism on extraordinary large scale.

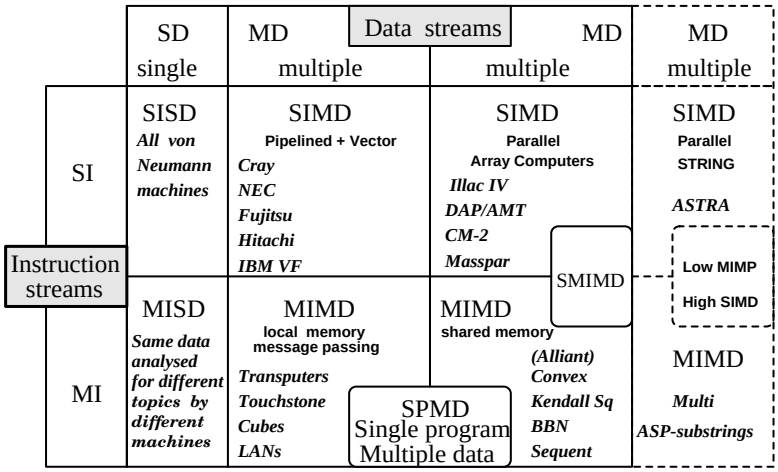

Fig. 1. Flynn's classification

There is, however, a frequent objection against the astronomical increase of the number of processors. In order to better understand this problem one should clarify the notion of "speed-up" as a quality factor making judgment about the merits of different architectures.

1.3 "Realistic" speed-up

It is the general assumption: Only part of a given algorithm can be parallelized. It is hard to debate this fact if one regards the algorithms in an abstract way. From this assumption directly follows the **Amdahl's law**: if an improvement is only applicable for a fraction f of time:

$$Speed\text{-}up = \frac{Old\ execution\ time}{New\ execution\ time} = \frac{1}{(1-f)+\frac{f}{N}} < \frac{1}{1-f}.$$

Despite the fact that N processor speeds up N-times the given f fraction, after some not too large value of N the sequential $(1-f)$ part will dominate. Table 1. shows that already relatively small N values almost saturating the speed-up limit set by the Amdahl's law.

f	N	$Speed-up$	Asymptotic $(N=\infty)$
0.5	5	1.66666	2
0.9	90	9.00000	10
0.99	990	90.0000	100

Table 1. Fast saturation of speed-up due to Amdahl's law

Due to the fact that it is hard to imagine a program with less than 1 percent sequential part it looks useless to build systems with more than 1000 processors.

In particle physics experiments, however, one is encountered with some specific circumstances which should be taken into account during the calculation of the "realistic" speed-up. The measurement process consists of the recording of individual particle interactions. When one interaction (= event) occurs, the whole detector is closed until the system recovers and becomes ready for the next event. During the execution of the experiment one tries to maximize the LIVE-TIME of the measurement relative to the DEAD-TIME required for the detector's hardware recovery. In this sense one can make an additional assumption:

> **If the sequential part of the data processing is not exceeding the hardware dead-time portion, then the parallelism is fully paying off, or in general:**

$$"Realistic"\ speed\text{-}up = \frac{Old\ execution\ time - DEAD\text{-}TIME}{New\ execution\ time - DEAD\text{-}TIME}$$

The above mentioned specificity, that **independent** collisions are studied, provides an other (rather trivial) way to beat the Amdahl's law: each event should be sent to an other computer. Thus the inherent "event parallelism" can be used to eliminate the sequential part completely: $f \Rightarrow 1$ (!!). This is also true for the event simulation.

Particle physics is not limited by Amdahl's law, thus we can bravely dream about millions of processors, but it is limited in MONEY.

1.4 General strategies for parallel algorithms

Having the beautiful massively parallel hardware one can be still far away from the real solution, because the selection and the implementation of the appropriate algorithm is not at all a trivial problem. For illustration one can regard some intentionally simplified border case strategies:

1.4.1 Direct parallelization

This is a specially attractive strategy for SIMD machines, because it requires minimal effort from the side of the programmer if he has already the sequential algorithm. It can be well demonstrated by the classical anecdotal case when the headmaster asks the teacher to find the "To be or not to be" quote in Shakespeare's Hamlet. If he is in a class, then the quickest way to find it would be to assign one page to each student (assuming large enough number of pupils) and give the order "scan your part and raise your hand, if you find the quote". Of course, the teacher could do the same himself regarding the whole volume as his part, but what a difference in speed !?

Unfortunately this strategy has only very limited value in practice, because most of the problems do not have this simple repetitive nature.

1.4.2 Tailor made hardware for specific algorithm

If the problem is so important that it is worth to design a tailor made hardware system for its solution one could really get an ideal arrangement. Let us demonstrate this in a definite example, the matrix multiplication:

$$C_{ik} = \sum A_{ij} B_{jk}$$

In case of sequential algorithm n^3 multiplications are executed by 1 processor. In massively parallel machine one can assign one processor to each multiplication, which is not totally crazy in view of our definition of really massive parallelism which requires for $n = 100$ "only" one million processors. Thus only 1 multiplication is required and even if it is much slower with the cheap processors, one can still have considerable gains. (The human brain is working rather well along this principles.) In case of ASP, which will be studied later in detail, this situation is realized, of course, only up to $n < \sqrt[3]{N}$, where N is the number of processors in the string. This is an example for the case, where the computation time is independent of the problem size if the machine is big enough. (In strict sense, if one takes into account the additions in the formula, one have to add a $log_2(N)$ term too.)

1.4.3 Optimal mapping

In real life the machine is never big enough, therefore it is the task of the programmer to search for optimal mapping of the problem to the given hardware. Considerable effort is made to develop super clever CROSS COMPILERS which would able to discover the generally hidden inherent parallelism within the problem and then able to exploit this by adapting it to the hardware limitations. Probably that programmer genius is not born yet, who is able to write this ideal compiler, but there are a number of characteristic parallel algorithms which can teach the way for "manual" realization of optimal parallelization. And it is a great pleasure to invent such parallel algorithms which doesn't have direct analog sequential realization, developing some sense of "parallel thinking" which can use instinctively the opportunities provided by the hardware.

2. ASSOCIATIVE "THINKING"

Associativity is an essential ingredient of human thinking. In computer science it is used mainly in the restricted sense:

"content addressing".

The (complete or partial) matching of the content in different parts of the computer system resembles to the thinking process creating logical connections between seemingly independent objects i.e. they become "associated". Newton's apple serves as a typical example for one of the most famous association. In this case Newton was, who realized the **common content** (namely, the gravitational attraction) which causes the free fall of the apple from the tree and the circular motion of the Moon around the Earth.

2.1 Associative memory

In most of the cases the content addressing concerns the bit content of the memory cells. In normal memories a unique address is assigned to each cell containing given number of bits (= word).This word content is read out during the access to the memory. The associative memory works in a reverse way. The searched content is presented simultaneously to **each** memory cell. One can have several outcomes:

a) NO-MATCH: no memory cell exists with the required content;
b) SINGLE-MATCH: only one cell is hit;

c) MULTIPLE-MATCH: the searched content is found in more than one cell.

Physically this means that each memory cell has an additional **comparator** unit. In more developed systems one can also search for partial content matching. This subset comparison is realized through ternary logics. The content addressing bus presents the common searched content for each memory bit separately in 0, 1 or *don'tcare* states. This comparator scheme with ternary input is illustrated in Fig. 2.

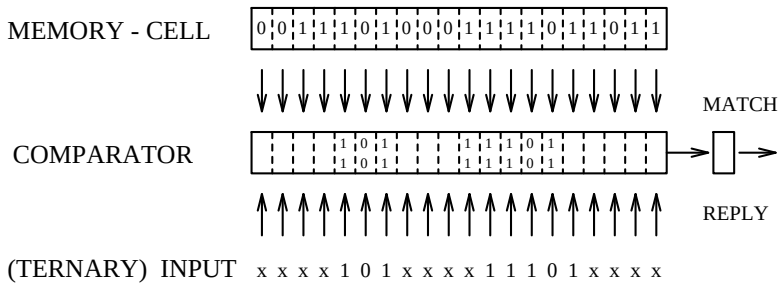

Fig. 2. Content addressable (associative) memory

Using this "subset" feature of content addressing one can realize the so called PSEUDO-ADDRESSING. Let us take an example: if one stores in the first 10 bits of an 32-bit associative memory cell the serial number of the cell from 0 to 1023 and stores the "real" information starting from the 11-th bit then content addressing in the first bits will be equivalent with normal addressing in a 1k memory. Of course, this use is rather uneconomical because almost one third of useful memory space is lost for addressing and one needs a comparator for each cell.

2.2 Associative phone-book

One of the simplest application of associative memory relies on the above mentioned pseudo-addressing scheme.Let us assign to each client a long memory cell which can contain the name, phone-number, profession and address (Table 2.). In a normal phone-book it is easy to find the phone-number if one knows the name, because they are printed in alphabetic order. It is even more simpler in our associative memory, where the names can play the role of pseudo-addresses replacing the serial numbers of the previous example storing the name in the first segment of the long associative memory cell. The only difference is that there can be "holes" between this type of pseudo-addresses, because there are no names for all bit combinations. Additionally to this, if we search for a person without telephone, there will be no correct match. Of course, there can be different persons with identical names. They produce multiple match-replies, but by taking the generally short list of matched cells line-by-line one can easily identify the searched person by help of the other attributes. Of course, all these possibilities are present in the normal phone-book, but it is really a pain if one has a phone-number and doesn't remember the person's name. Even in computerized version of the phone-book this reversed task is

rather time consuming in a sequential machine with normal memory, because in average one should read one half of the total memory to get the answer for an existing client and one should waste the full reading for the single bit "NO" information if there is no such number in the book. In contrast, the associative memory gives the exact answer in a **single step** (or in few steps of multiple-match, if more than one person is callable on the given number). One can also easily find a quick answer for such type of questions:

Is there any dentist in the Main street, who has a phone?

name	profession	address	number
ABA ABA	physician	Main str. 7	117 336
ABA BEA	teacher	Fleet str. 4	342 752
ABA EDE	miller	Rakoczi ave. 38	462 781
ABA IDA	engineer	Penny str. 15	279 347
ABA ÖRS	dentist	Magyar str. 76	972 413
ABA TAS	lawyer	Sunset blv. 20	362 478
ABBA EVA	hairdresser	Museum sqr. 6	784 652
...			

Table 2. Universal phone-book

2.3 Associative tree traversal

By simple use of associative memory one can easily solve formal mathematical problems too. In case of the tree traversal task one should identify all the ascendants of a given node in a tree-graph. The Node and Parent names are assumed to be stored pairwise in an array in the memory cells of the computer.

If you want to find all the direct ascendants of node H in the graph shown in Fig. 3., you first have to **scan the array sequentially** until you found Node name = H. This gives the first parent, G stored in pair with H. Then you need to **scan the array again** from the beginning until you reach Node name = G. This gives you H's grandparent. You **continue scanning** in this way until you reach the root (a node with Null parent). The resulting family list from H upward is: H, G, C, A.

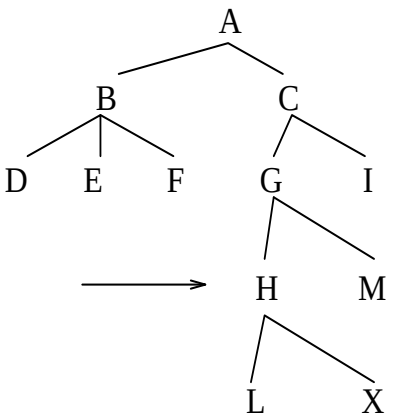

Fig. 3. Tree-graph

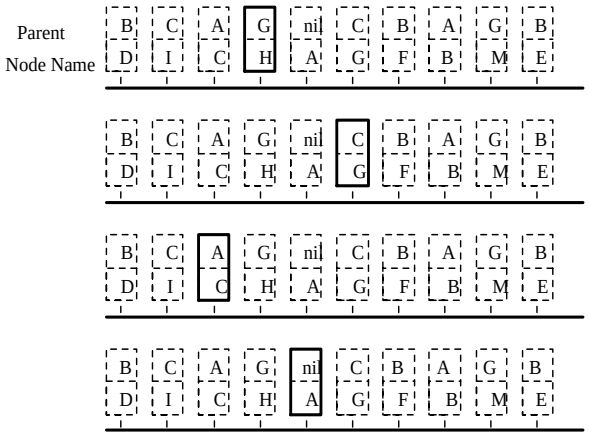

Fig. 4. Associative search

To implement the same tree on an associative machine, you would allocate a memory cell to each node of the tree. In a similar way as before, each associative memory cell would store the node name and the name of its parent . The main difference is that we have content addressing (i.e. associative access) to find any node we want.

Again, imagine we want to find the ascendants of node H. Instead of scanning all the elements one after each other to find the one containing H, the corresponding cell accessed **directly**. Knowing the cell, the rest of its content provides the first parent, G. Next we **go straight** to the element storing details of node G. Thus the second step gives the grandparent. And so on... Thus the number of accesses needed to complete the task in this way is the same as the number of node levels (in this case it is 4). See Fig.4.

Using the sequential method requires many more accesses to the data structure. The exact number depends on the ordering of data in the array. If by chance one starts from the lowest level, then the best case is finished after 15 accesses (e.g. 1 to find L, 2 to find H, 3 ....), and the worst after 45 (11 to find L, 10 to find H, 9 ....).

2.4 Associative string processor

In order to create from an associative memory an associative processor it requires only a simple step. Adding a 1-bit processor unit to each memory cell where already the comparator itself represents a primitive logical unit, one can modify the content of the memory in a controlled way simultaneously in each cell depending on their original content. This modified content will react for the next content addressing instruction in a different way. By clever programming this can lead the way toward the solution of the given problem.

The above system is becoming really flexible when one allows communication between these associative processors, that is depending on the content of a given cell one can influence the state of an other cell. The simplest possible communication between left and right neighbours creates the 1-dimensional string like structure. This Associative String Processor architecture concept was worked out in details by prof R.M. Lea and his collaborators in Brunel University [5]. Despite of its conceptual simplicity this architecture is rather flexible and provides ample opportunity to solve very complex problems. It turns out to be specially effective in the case of so called iconic to symbolic transformation.

2.5 Iconic versus symbolic method

In particle physics and in many other areas it is a frequently occurring situation what we can call the **sensor/response** paradigm. Human senses (especially the eye) or technical monitor systems collect the information from their environments according to some ordering in space and/or time. If one maps this information into a computer system in such a way which reserves the basic inherent relations between the data units in a faithful way, one speaks about "iconic" representation, because most frequently this corresponds to some kind of 2-dimensional image or series of images. According to the sensor/response paradigm, in a well developed system after massive data reduction and feature extraction one derives the symbolic representation producing the list of objects, providing the formal base for the decision and response process. In big HEP experiments for example this appears as the **trigger-decision** paradigm shown in Fig. 5.

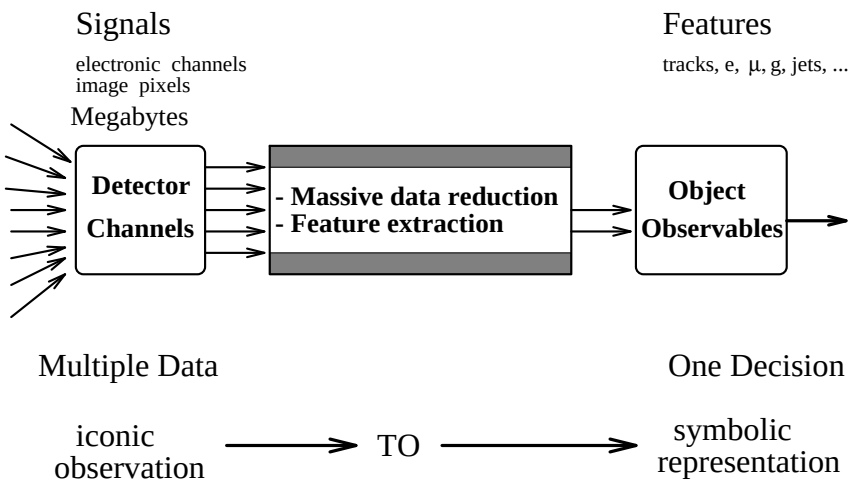

Fig. 5. The trigger-decision paradigm

One can, however, observe similar effects in more modest systems too, where the comparative advantages of the iconic versus the symbolic representation can be explained more directly. Fig. 6.a shows a part of a streamer chamber photograph in "iconic" representation. Fig. 6.b gives part of the numerical "symbolic" (x,y) list representation, which was created by a simple "zero-suppression" data reduction algorithm.

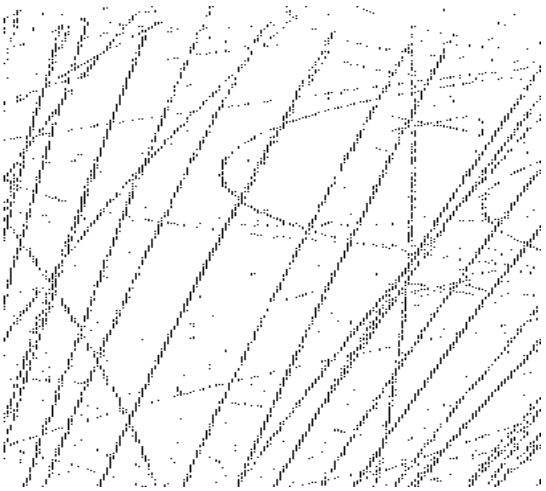

Fig. 6.a "ICONIC" representation

(101,107) (101,178) (101,227) . . . (101,875)

(102,108) (102,131) . . .

(107,104) . . .

.

.

.

(579,102) . . . (736,914)

Fig. 6.b "SYMBOLIC" (x,y) list representation

It is obvious that the symbolic point list requires much smaller memory space for storage, but one looses or better to say hides the information whether a given point belongs to a real track or to the background, or whether 2 points are belonging to the same track or not.

It depends on the given problem which representation provides the most effective algorithm. In case of particle track reconstruction the symbolic list method is easy if there are only few coordinates on the list. For dense, complicated images the symbolic representation has the disadvantage that the "trivial" topological relations, which are obvious on the detector level, are lost.

One can however exploit very effectively the "topological associations" of elements on the "raw-image" i.e. in the iconic representation applying the following techniques:

a) simple **neighbourhood** algorithms can be used (cluster search, track following etc.);

b) **whole picture** could be processed by parallel evaluation of each pixel's surroundings (so called locality of reference);

c) by **careful mapping** of the images the "associative effects" can be amplified.

All these techniques call for computer architectures which are well adapted for "non-numeric" calculations. Exactly this is the domain where the Associative String Processor (ASP) may beat the other architectures. In ASP there is no need to translate the sequential algorithms into parallel ones, because one should run on ASP machines conceptually different new algorithms, and it is the pleasure of the programmer to invent the most appropriate ones corresponding to the given hardware.

3. ALGORITHMS

The power of associative thinking can be demonstrated only by good examples of convincing, simple and hopefully interesting algorithms performing the iconic to symbolic transformation (at least conceptually) in a transparent way. In order to emphasize the general nature of these algorithms we shall rely only on the basic features of the Associative String Processor minimizing the references to any given realization which will be discussed in subsequent sections. These basic features are:

i. associative memory cells;

ii. 1-bit CPU with some 1-bit registers;

iii. string communication between left and right neighbours.

3.1 Associative football referee

In the year of the '94 Soccer (Football) World Championship in the USA it seems to be evident that one remembers the difficult task of referees on the judgment concerning the OFFSIDE situation. If there would be an on-line automatic electronic referee as in many other branches of the sport all the debates could be avoided once for all. Here we try to show that this is a very easy problem for an associative processor, if one treats it on the iconic representation.

Let us assume there is a CCD camera far above the field whose resolution is good enough to distinguish objects of size about 10 centimeters. For definiteness we say that to each 10*10 cm^2 piece of the field corresponds in the CCD image a pixel. The basic constituents of the game are coloured in different ways: field is **green**, the lines are **white**, the players of the A-team are **red**, the players of B-team are **blue**. (The model can be complicated by more details but hopefully this is enough to understand the essence of the algorithm).

If the camera is able to produce 1 image per millisecond (1 ms = 10^{-3} second) then between 2 images the maximal displacement of the ball will be less than 4 centimeters, assuming as a maximal flying speed less than 144 km/hour = 40 m/sec. It corresponds half of the pixel size, therefore it is completely adequate to evaluate the situation by this frequency. Thus our electronic referee will produce a decision in every millisecond, of course, it will disturb the human referees only in case of the OFFSIDE, thus the humans can turn their attention to other aspects of the game.

The key point of the algorithm is the **iconic mapping**: each pixel has its own COMPUTER. It looks frightening to demand many thousands of computers for this purpose, but don't forget they are simple and cheap. Nowadays one can by 64 Megabits on a single normal memory chip, and the associative processors are "merely" intelligent associative memory cells.

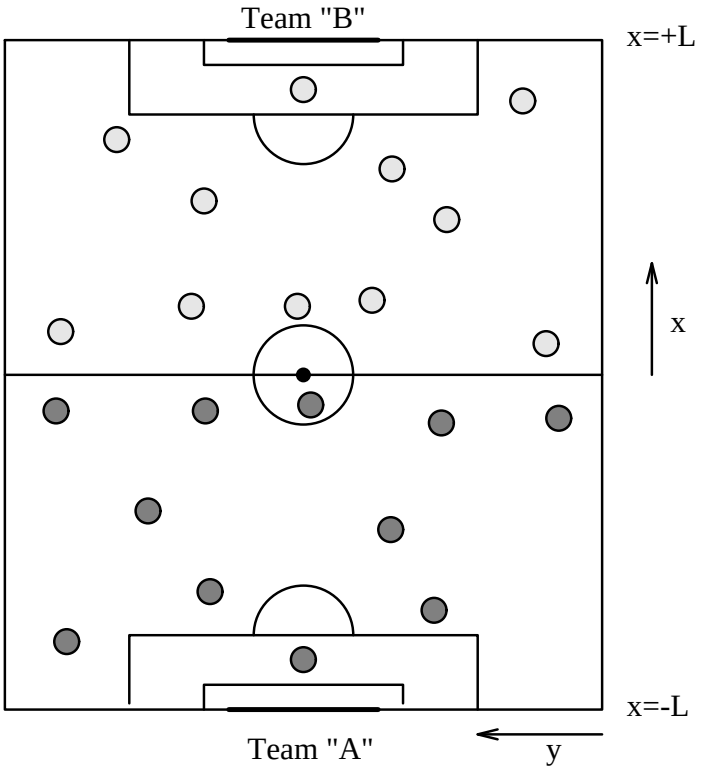

Fig. 7. Soccer field

Introducing the coordinate system according to Fig. 7. the string of processors will be filled along the horizontal lines at the given x values, starting from $x = -L$ finishing by $x = +L$. That is team-A intends to direct the ball to goal at $x = -L$.

In any moment one can identify four basic states as it shown in Fig. 8.:

a) Ball is free: [0 , 0];

b) Ball belongs to team-A: [A , 0];

c) Ball belongs to team-B: [0 , B];

d) Ball belongs to team-A **AND** team-B: [A , B].

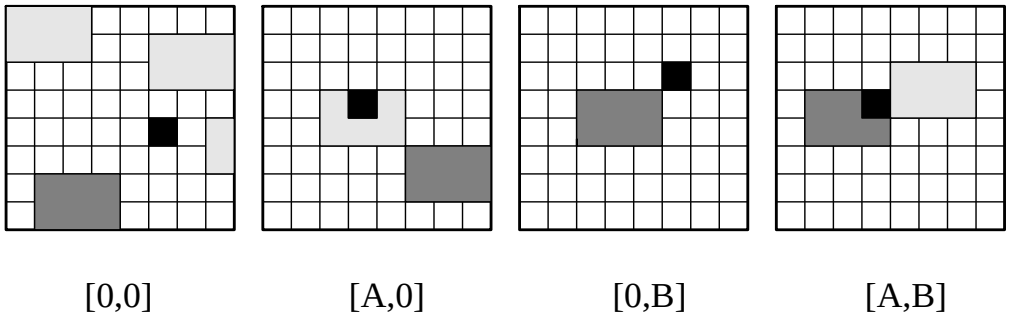

Fig. 8. Ball states

We intentionally left out the case when the ball is below some player and assumed if they are in the same pixel, then the ball is always above the player.(As the inclusion of this scenario is also well adapted for an associative algorithm, but for sake of simplicity we leave its realization to the intelligent reader.)

In this model realization we test the OFFSIDE rule formulated in the following way:

" Any player between an opponent's goal and last player (unless he was followed the ball) is OFFSIDE and out of play." (Private comment: the goal-keeper doesn't count.)

Reformulation of the rule in ASSOCIATIVE LOGICS:

If the ball is in the moment $time = t_n$ in state $S_n = [0,0]$, but it was in moment $time = t_{n-1}$

A: in state $S_{n-1} = [A,0]$, then team-A is on OFFSIDE if:

$$min[x_A] < min[x_B],$$

where the minimum function doesn't include the goal keeper assuming some distinguishing mark on him.

B: in state $S_{n-1} = [0,B]$, then team-B is on OFFSIDE if:

$$max[x_A] < max[x_B],$$

where the minimum function doesn't include the goal keeper assuming some distinguishing mark on him.

C: There is not OFFSIDE in cases if S_{n-1} = [0,0] or [A,B].

Knowing that the size of the player is always larger than the ball size the calculation of state S_n is rather straightforward by content addressing. If there is a "ball contact" e.g. with team-A then there will be at least one orange pixel which has a red pixel in its direct neighbourhood.

Please realize the elegance of this approach: the algorithm is completely independent of the image shape of the player (whether is he running, standing or lying in the given moment), where are his arms or legs (if they coloured correctly). Here the topological content addressing gives the answer **without any calculation**. It is also irrelevant that how many cells are occupied either by the ball or the player, because the neighbourhood test will be executed by each associative processor parallelly which has the "ball" content, and all the ball containing cells will give a common report by OR-ing the results of individual searches, thus the so called single "match-reply" will identify the ball situation uniquely.

Using the stored value of S_{n-1} one can compare it with S_n. If one identifies case A or B, then calls for min or max routines, respectively, otherwise just stores S_n for next time to be used as S_{n-1}. These min-max algorithms relies on the x-coordinates of the players. By the above mentioned "pseudo-addressing" scheme one can pre-store in each processor's associative memory the corresponding x value because it is fixed during the match assuming that the camera sees the whole field without any movement. This is a 1-dimensional problem: one should search for the first and last red or blue cell and read out the corresponding x value. Of course, as in case of ball-contact routine, it is irrelevant what is the concrete image shape of the players. Thus the decision can be made really by the shortest time delay, because after 2 parallel searches one should make only one single numerical comparison. This can be executed much faster than 1 msec, the technical challenge is not in computing but in the detection hardware, whether the CCD interface is fast enough to load the images into the ASP computer in time.

It is worth to mention that if one would have a camera with better resolution (and have correspondingly more processors) the algorithm would be identical! That is this algorithm is SCALABLE.

3.2 Cluster algorithms

In HEP experiments the particles produce clusters in a number of detectors: calorimeters, CCD cameras, TPC's etc. Sometimes for a fast decision one is interested only in the number of "good" clusters; sometimes their shape is important - sometimes not; sometimes one needs to collect all the information which belongs to the cluster or it is enough to find some characteristic point in it etc. In the following we try to present some demonstrative examples for different types of basic cluster algorithms.

As in the soccer case we assume that the data are presented in 2-dimensional iconic form, mapped line-by-line along the string and each pixel has its own processor except the cases when it is stated explicitly otherwise. We define a cluster as a connected subset of pixels which are touching each others at least by their corners.

3.2.1 Cluster classification

It is a frequently occurring case in HEP that the real particles produce clusters which obligatorily have at least 2 active detector channels e.g. in calorimeters the induced showers are extended over several detector cells. In contrast to this the randomly occurring thermic or electric noise produces signals in an uncorrelated way mainly in individual single channels. It would be very useful to get rid of this noisy background clusters as fast as possible. In the iconic representation this task means a cluster search algorithm which identifies the single pixel clusters. It is simple enough to start our image processing lesson.

Before going into the details of the algorithm, we introduce the notion of ACTIVITY-BITS. These are special 1-bit registers in the CPU attached to each associative memory cell, where the iconic (in this case only black-and-white, binary) image is stored. These are denoted by A1, A2 ... A6 and are also content addressable as the memory cells. The algorithm itself consists of two phases: preprocessing and decision which are illustrated on a small representative example in Fig. 9. **A:** Preprocessing

1. LOADING of the binary image into the IMAGE-BIT of the memory;
2. MARK the A1 activity bit in those processors where the IMAGE-BIT is equal to 1;

3. SHIFT "LEFT-RIGHT" the content of the IMAGE-BIT and store 1 both in the effected processor's A2 activity-bits **AND** memory IMAGE-BIT. Please remember: this move will destroy the original image, but we were careful to preserve a copy in form of the A1 activity-bits;

4. SHIFT "UP-DOWN" (it means shifting by the line length) all the corrected IMAGE-BITS, but in this case store them only in the A2 activity-bits. Please remember: this is an OR-ing procedure if in a given processor A2 was already set to 1, then it remains in that state, only those A2 bits will be effected by the subsequent setting which had 0 previously.

The essence of this procedure that if a pixel has a neighbour then by the shifting exercise its A2 activity-bit will be obligatorily set. In case of single cluster pixel there is no neighbour which could be smeared out to cover it.

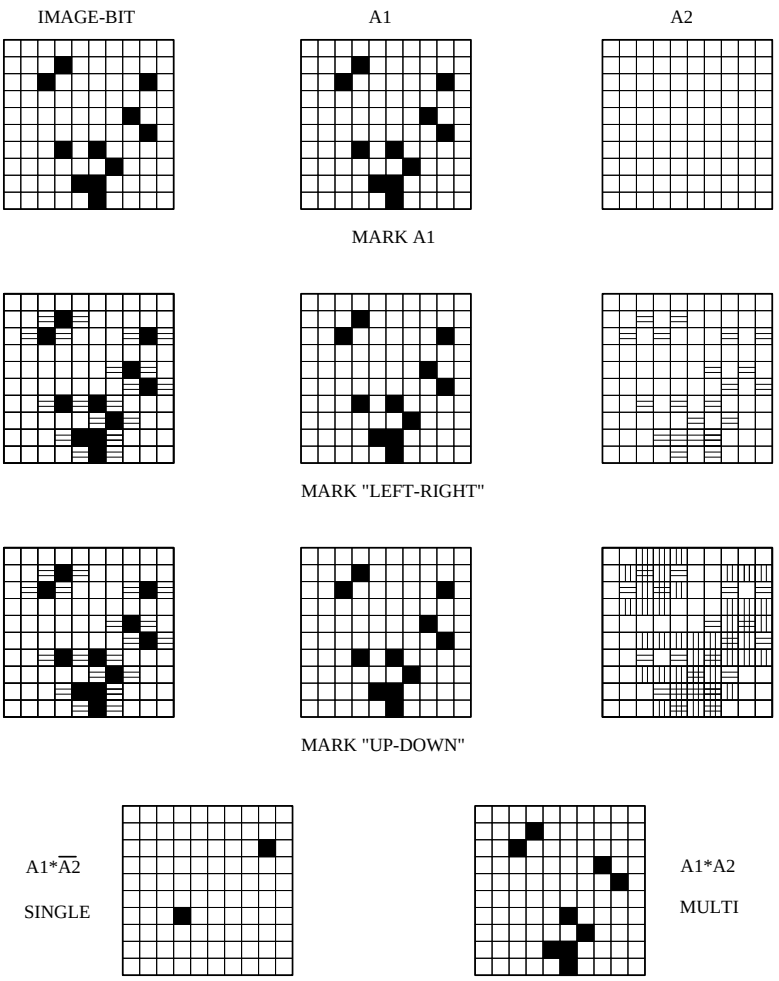

Fig. 9. Single cluster selection

B: Decision

1. SINGLE (background) cluster selection:

One sets to 1 the IMAGE-BIT in all processors where

$$A1 \cdot \bar{A}2 = 1$$

otherwise set to 0.

2. GOOD (multiple) cluster selection:

One sets to 1 the IMAGE-BIT in all processors where

$$A1 \cdot A2 = 1$$

otherwise set to 0.

Of course, it is our choice which decision will be really executed. The second decision will produce the background free image containing only the "good" clusters.

3.2.2 Cluster counting

In case of experiments where the number of cluster provides the key information for the decision (e.g. one can distinguish K_S and K_L kaon decays by the number of photons emerging from the decay chain: 4 and 6 respectively) it is crucial to have the previously

described fast cleaning algorithm, but we should have an equally fast counting algorithm too. The basic idea is to reduce by an universal algorithm (universal means independent of cluster size and shape) each cluster to a single point and then count the surviving single point pixels in the image. First we start with an extremely simple algorithm then by gradual improvements we reach the satisfactory version.

A: POINT killer method

By using the same technics described above each pixel which contains 1 in its IMAGE-BIT kills the eventual hits (i.e. "1"s) in four neighbouring directions: left, left-above, above and right-above. Illustrative examples are shown in Fig. 10. In Fig. 10.a one sees that the not vetoed surviving pixels really achieves the correct reduction. But this method fails, for instance, in cases like shown in Fig. 10.b, where there is a "double step" on the cluster's down edge. Instead the correct 1 we get 2 clusters.

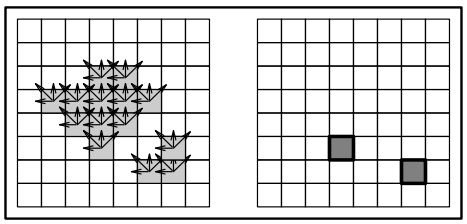

Fig. 10.a Good

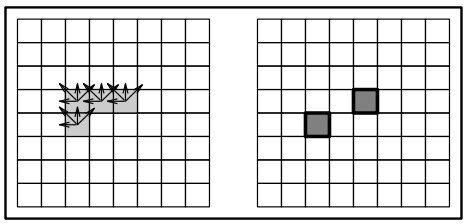

Fig. 10.b Bad

B: INTERVAL killer method

The above problem is avoided by using **intervals** instead of points. Intervals, which are represented by a contiguous string of neighboring horizontal hits, are well adapted to the ASP line (string) architecture and represents the most natural generalization of the single processor concept to a bunch of consecutive processors along the communication line. The basic steps of the procedure are:

. Define INTERVALS in line i;

.. INTERVAL is killed if it touches at least one hit of an other INTERVAL in the "down" line $i+1$;

... COUNT the surviving INTERVALS.

Though this routine cures the basic problem of the killer method for "convex" clusters, it fails in the more sophisticated "concave" cases shown in Fig. 11.b.

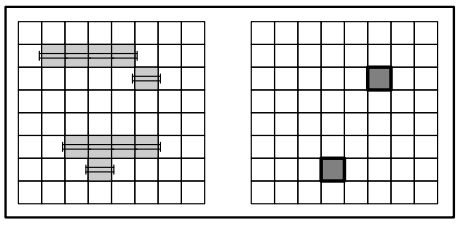

Fig. 11.a Good

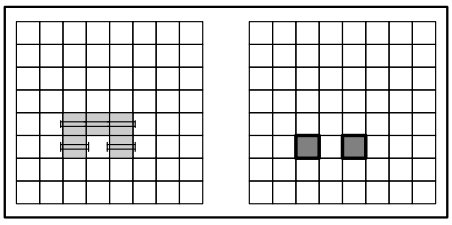

Fig. 11.b Bad

C: Single-hole INTERVAL killer method

In HEP the size of the clusters is generally not so large, therefore one doesn't expect excessively extended "concave" structures, i.e it is enough to take care of single gaps. With a slight modification of the algorithm one can **fill** these single "concave holes". This filling however will blow-up the whole cluster by one pixel in each side on the horizontal direction, therefore in order to avoid cluster merging we should assume that there is at

least 2 empty pixels in line i between points belonging to different clusters. Fig. 12. shows the modified procedure, where the bridging step was added as a first step redefining the surviving INTERVALS.

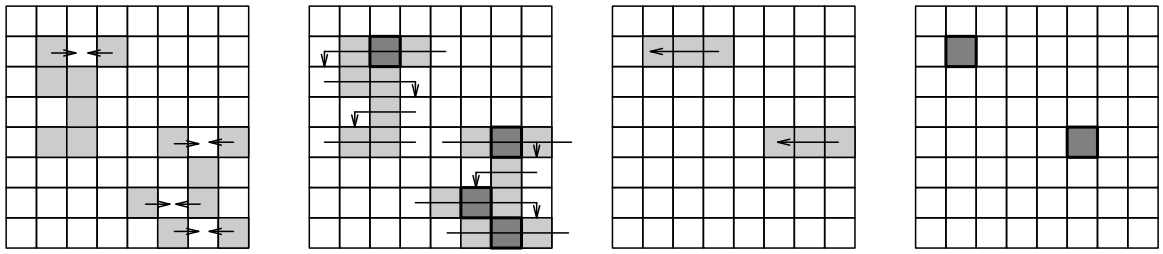

Fig. 12. Corrected interval killer method

It is worth to remark that already the original INTERVAL algorithm is unsensitive the vertical column-type non-convexities shown in Fig. 13. As a last remark that one can add to the cluster counting topic is that it can be made extremely fast by avoiding the line shift during the smearing process (which requires as many steps as long is the line) if one makes double mapping into the ASP. By appropriate interface one can load the content of 2 lines in each pixel line, because there is enough bandwidth. In this case the execution time of the cluster counting algorithm will be completely independent of the size of the image, that is the scalability will hold exactly.

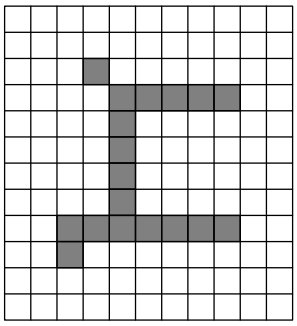

Fig. 13

3.2.3 Convolution

So far we dealt with the clusters as "binary objects": signal/noise or one cluster one count. In a number of cases some kind of global characteristic of the pixels of a cluster is carrying the relevant information. For example in calorimeters the cluster energy sum can be regarded as such a global parameter. If one knows the approximate size and shape of the cluster then summing up the energy content of the individual pixels belonging to a given cluster in the "middle" cell one gets the total cluster energy. Some simple frequently used cluster shapes are shown in Fig. 14.a and b. This energy sum procedure represents a special case of convolution which is generally defined by the following formula:

$$C_{i,j} = \sum b_{m,n} A_{i+m,j+n} \qquad -k \leq m,n \leq k,$$

where the $b_{m,n}$ kernel is shown in Fig. 14. for the energy sum in cases $k = 1$ and $k = 2$.

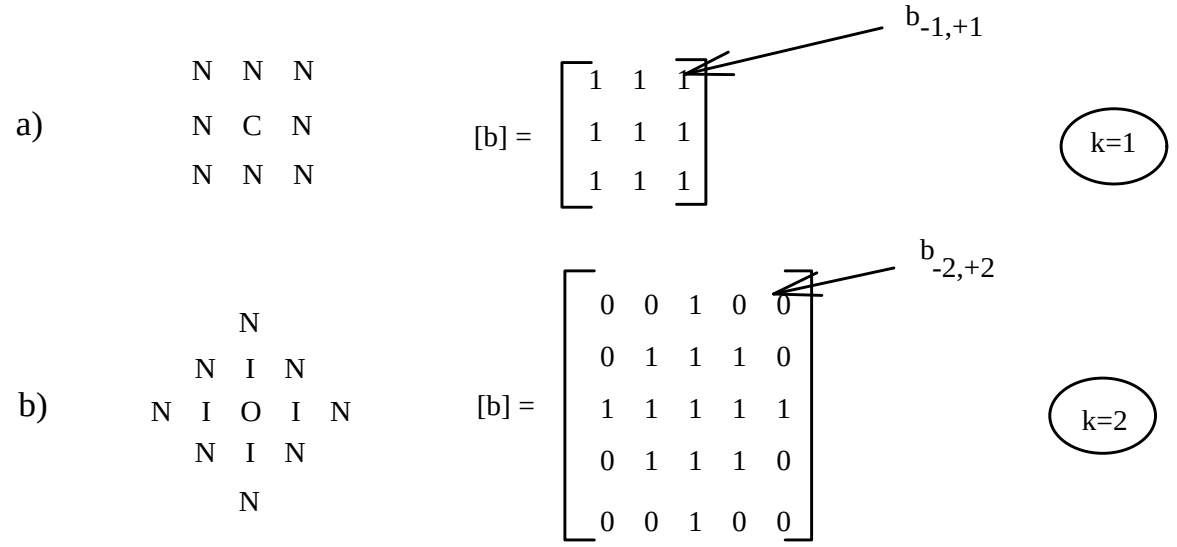

Fig. 14. Peak kernels

In sequential calculation the most effective way is first identify the cluster center and then collect the sum from the surrounding cells. In case of parallel computing one can reverse this order. Each cell can assume about itself that it is a cluster center and collecting the sum it can execute the calculation in parallel once for all. The real cluster centers will be distinguished by the fact that they really will contain the full content of the cluster, therefore the accumulated sum in them will show a local maximum.

The general convolution algorithm can be executed by standard SHIFT and ACCUMULATE commands in a very straightforward way which doesn't profit really from the associative character of the machine, therefore we don't discuss it in details.

3.2.4 Peak (maximum) finding

In most frequent cases in calorimetry the most interesting cluster is the one which has the highest energy. Therefore it has a special interest to find the pixel with maximal memory content which after the eventual convolution corresponds the energy of the highest energy cluster. We illustrate the associative logics involved in this procedure on a simple numerical example.

Initially one assumes that any of the integers shown in Fig. 15.a could be the maximum. Think of these integers as being in a set called "potential maximums". We need to reduce this set in some way until only the maximum value remains.

The reduction of this set is performed by a particular test. Those elements that fail the test are no longer considered, i.e. are removed from the set of "potential maximums". Repeat this test until the true maximum is found. Picture the elements of the array as binary numbers (Fig. 15.b) on such a way that the highest bit is at the top.

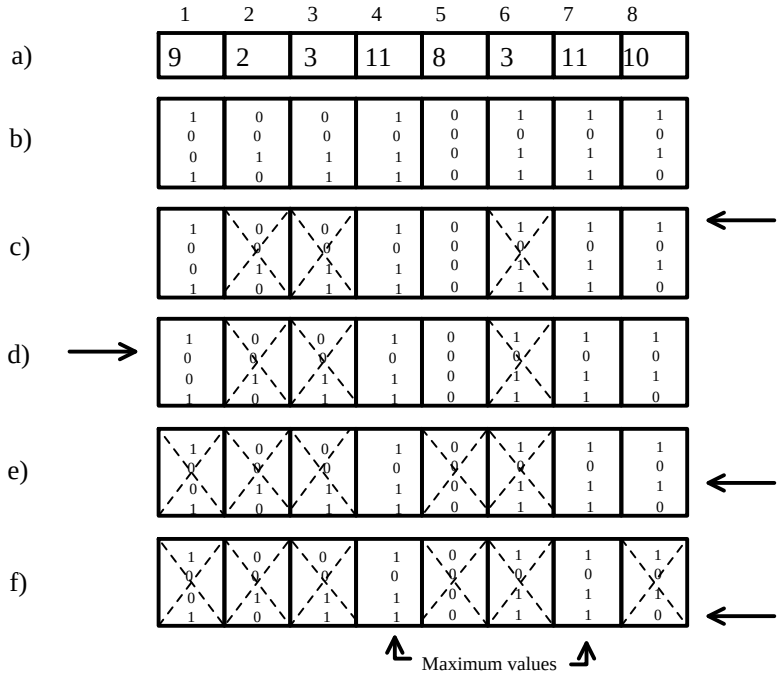

Fig. 15. Maximum finding

In parallel for each element in the set, we point to the top-most bit and perform our test. This is simply:

"Do you contain 1 at the current bit position?"

The processors with answer "NO" are removed from our set of possibilities (Fig. 15.c) and are not subject to future testing. Then we consider the next bit position, and perform the same operation upon the remaining elements in the set (Fig. 15.d). **NOTE:** If **all** elements answer "NO", then no action will be taken, the algorithm continues to the next step by setting up the new "current bit position".

This procedure continues until every bit position has been tested (Fig. 15.e). Of course, the number of bits should be known from the beginning because one has only limited space in the memory for storage of the tested numbers. The maximum element(s) are the ones which answer "YES" to all test (except when all elements of the set contain 0, as it was noted above).

A similar algorithm can be used to find the minimal value.

It is also worth to remark that the speed of this algorithm doesn't depend on the number of elements, but on the number of bit positions to be tested. So the process will be performed just as quickly with a million elements as will with ten.

3.3 Tracking

Most of the time the particle physicists are dealing with particle trajectories, the so called tracks. There are a number of tracking devices in use. One example for the output of such detector was shown in Fig. 6.a. In general this is a 3-dimensional problem, but most of the cases the pattern recognition is performed only in 2-dimensional projections dictated by the detector architecture. For sake of simplicity here we restrict our task also for that case. 3.3.1 JACK-POT algorithm: TRACK-CODE

So far we studied only such algorithms when the iconic mapping was realized by packing only one item per processor. Sometimes this may not be optimal. The next algorithm is a good example for the statement:

Correct mapping of the data provides already half of the solution.

Let us take multiwire proportional chambers (MWPC's) positioned at given z values, which are measuring the x-coordinates of the tracks emerging from some interaction point on the target plane. A typical arrangement is shown in Fig. 16. The vertex position (the interaction point) can be anywhere in x with some probability on the $z = z_{TARGET}$ plane.

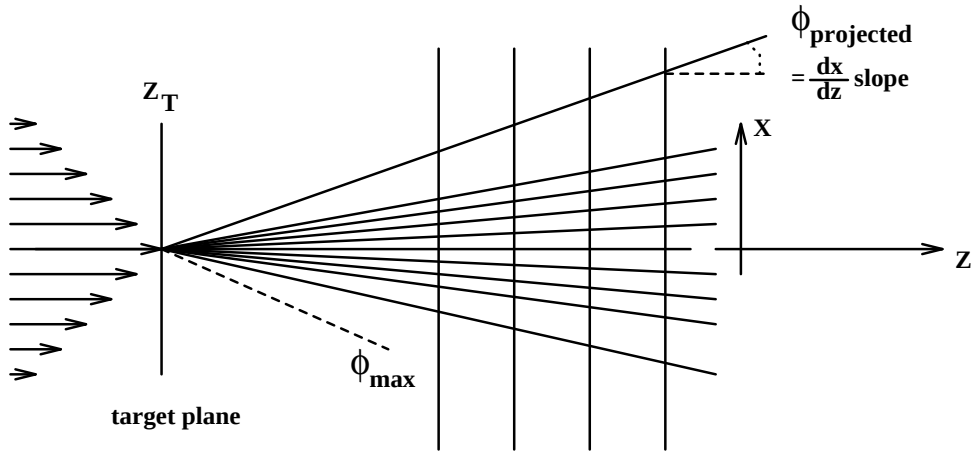

Fig. 16.

The points crossed along the trajectory are described by two numbers: [plane number# i ; wire number# k] , when the k-th wire was hit in the i-th chamber at $z = z_j$. What is seen on the graphic image, it should be mapped on to the 2-dimensional

BIT-ARRAY

in a topologically faithful way. In the associative memory consecutive bits can be grouped together in short strings called "serial field". Let us choose the length of this serial field to be equal to the number of the MWPC planes, N. Thus one can define the i-th bit of the k-th processor and by simple, one is tempted to say automatic assignment one gets a very natural mapping:

$$HIT[plane\#i; wire\#k] => BIT[processor\#k; memoryposition\#i]$$

This mapping requires as many processors k=1,2,..,M as many wires are read out in the chambers. Please note the index reversal which implies the physical "crossing" of chambers and processors, which is the essence of the algorithm illustrated by the mapping procedure shown in Fig. 17. Due to this arrangement we can play for the JACK-POT. The origin of this idea is the following: one can hit the JACK-POT on the "one-armed bandit" slot machines if after spinning the wheels they stop by showing identical fruits. In our scientific case we also have a JACK-POT track combination. If the track is parallel with the z-axis, then in each MWPC plane the same wire k^* will be hit. In this case the content of the serial field containing the JACK-POT track will be 11...1, which could be identified by a simple **content addressing**. Thus due to the clever mapping one gets without any programming **all** the tracks flying out off the target with angle $\theta = 0$.

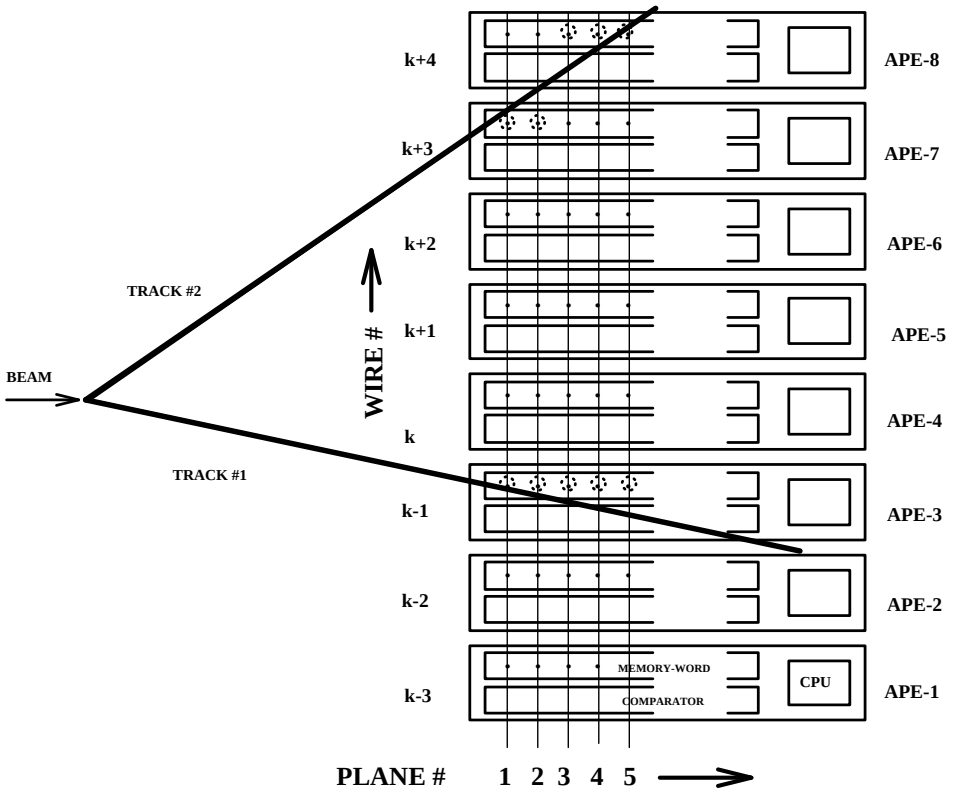

Fig. 17.

We are interested, however, not only in the existence of the track but its exact coordinates too. Using our old trick of "pseudo - addressing" one can define a second serial field in the associative memory of each processor containing the wire number, which is known by definition, just should be preloaded before the measurement. In this simple case, of course, the k^* wire number coincides with the serial number of the processor along the string.

All these parallel JACK-POT tracks has a common feature. If one denotes in general case the hit wire coordinates by W_1, W_2, ..., W_N then one can calculate

$$C_1 = W_2 - W_1; C_2 = W_3 - W_2; ...; C_{N-1} = W_N - W_{N-1}.$$

Independently of $W_1 = k^*$ one gets the vector

$$[C_1, C_2, ..., C_{N-1}] = [0, 0, ..., 0]$$

for each JACK-POT tracks. This C vector is called TRACK-CODE which is characteristic for all parallel tracks at a given θ angle. Tracks with identical TRACK-CODE differ only in the W_1 crossing coordinate in the MWPC plane# 1.

If one identifies the bit-columns of the ASP with the spinning wheels of the "one-armed-bandit", then by a lucky rotation one can reduce any TRACK-CODE to $[0, 0, ..., 0]$. Due to the fact that ASP is not a simple bandit, one doesn't need to trust in good luck, but should commit a systematic premeditated crime. The number of reasonable TRACK-CODES are limited and can be precalculated, then they can be (at least partially) ordered, thus it is enough to spin the wheels mainly in one direction only. For instance TRACK#2 is transformed to JACK-POT position if one shifts the columns#3,#4,#5 by 1 step. After hitting the JACK-POT the matching processor(s) give(s) the value of k^* and remembering the number of rotation "clicks" one can deduce the θ angle too.

The execution time of this algorithm depends only on the allowed number of TRACK-CODES which is defined by the physically predetermined θ range. It is independent of the number of tracks.

Just for summary it is worth to repeat the essence of the algorithm:

a) MAPPING different MWPC's in the same processor;
b) SHIFTING instead of CALCULATIONS.

3.3.2 MASTER-POINT algorithm

Here we should like to demonstrate the elegant efficacity of the iconic methods on an other aspect of track reconstruction. In an ideal world one could use unlimited number of MWPC planes in order to get maximal resolution. In practice, one should make a compromise, arranging the limited number of planes (allowed by the budget) in so called "super layer structure", containing a number of "stations" with closely packed subset of chambers (Fig. 18.). The pattern recognition strategies relies on the fact that inside a given station the change of the coordinate value is rather limited and the track pieces are so linear that the hits belonging to a given particle are clustering around a central coordinate value. Thus one can replace them by a single point the so called MASTER-POINT.

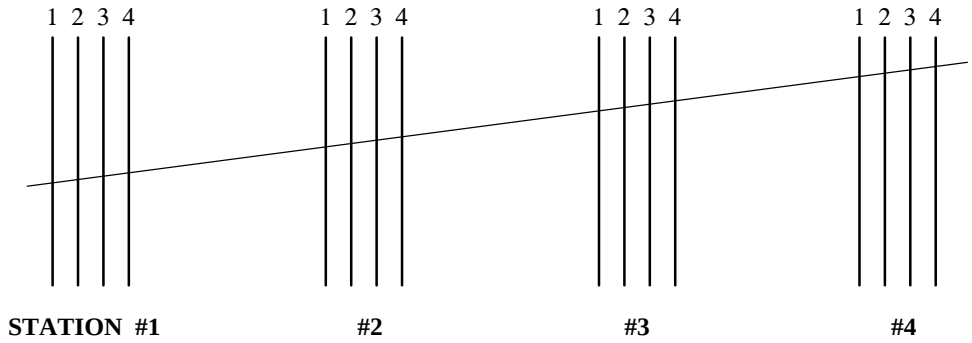

Fig. 18. Super layer structure

The other advantage of the MASTER-POINT idea that it can help to cure the unavoidable detector deficiencies like

a) inefficiency which is the probability that there is no hit registered despite the passage of the particle;
b) multiple hits can occur in the neighbouring wires despite the fact that only one particle was passing by;

c) noise is generated at some level in any system, that is one gets hit without particle.

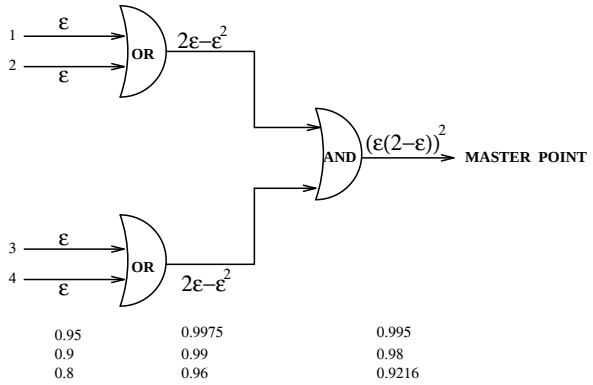

Fig. 19. Iconic average logics

Fig. 20. Master-point processing

For sake of definiteness we assume that each super layer consists of 4 MWPC planes. In this system the MASTER-POINTS are produced by the ICONIC AVERAGE logics shown in Fig. 19. The inefficiencies are taken into account by smeared "OR"-ing layers (#1,#2) and (#3,#4), respectively. The "AND"-ing of these smeared coordinates recenter the cluster and after clipping the edges the cluster is reduced to a single hit in most of the reasonable cases. This process is shown in Fig. 20. for tracks with positive inclination. In case of negative inclination the smearing shifts are executed in opposite order.

Assuming hit detection efficiency ϵ_x per plane, the MASTER-POINT detection efficiency will be:

$$\epsilon_{MASTER} = (\epsilon_x^2 + 2\epsilon_x(1-\epsilon_x))^2$$

For $\epsilon_x = 0.9$ (0.95) one gets $\epsilon_{MASTER} = 0.98$ (0.95) . Due to the fact that valid TRACK-CODE requires all the 4 MASTER-POINTS, the total track-finding efficiency will be

$$\epsilon_{TRACK} = \epsilon_{MASTER}^4 = 0.92(0.98)$$

Thus the MASTER-POINTS are defined as "centers of gravity" by shifting (in order to take into account the eventual x_i jumps) and by bit-logics operations completely parallelly. The execution is independent of the number of wires per plane and the number of particles traversing the super layer.

3.3.3 Hough-transformation

Though the Hough-transformation was not invented for associative parallel processors it can have very interesting realizations on this architecture. The essence of the method: the track reconstruction is transformed to peak finding, that is one should apply in the final search a clustering algorithm instead of a tracking one.

A line (or in more general, a curve) in Cartesian coordinates is represented by a single point in the so called "parameter space". If the points (x_i, y_i) belong to the line which passes at a shortest distance d from the origin of the coordinate system and which has a normal with an angle θ relative to the positive x-axis then the value of the expression

$$x_i * cos(\theta) + y_i * sin(\theta)$$

is independent of i and always gives d (Fig. 21.). For the opposite case when (x, y) point is not on this line the value will be different. Thus calculating for each "black-pixel" in the image for different θ_1, θ_2, ..., θ_k, ... values the corresponding d_1, d_2, ..., d_k, ... and filling a 2-dimensional (d, θ) histogram, the lines from the original image will stick out as peaks on this "lego-plot". The problem of LINE-SEARCH in the (x, y) plane is transformed to PEAK-SEARCH on the (d, θ) Hough-transform plane. This form of Hough-transform is ideal for symbolic treatment in sequential machines or for data driven pipelining processors.

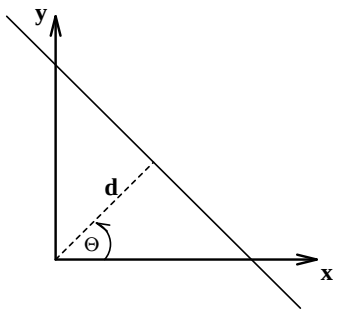

Fig. 21. Hough-transform parameters

For the iconic approach, however, one should rephrase it in order to avoid the lengthy floating point calculations and reach a form which conserves the topological relations between the pixels.

To each (d, θ) bin of the Hough-transform corresponds a corridor on the original image. Fixing θ but running d one gets parallel shifted copies of the starting corridor. Calculating this "$\theta - column$" of the Hough-transform is equivalent to counting the number of "black-pixels" in each of these corridors. In the proposed ASP algorithm the trick is that one is able to define this corridors in a natural way on the original image itself.

The Single Angle Hough-transform (S A H) procedure is based on the idea of A. Krikelis. In the "iconic" representation of the 2-dimensional image each PIXEL has its own COMPUTER. Comparing to the previous more "economic" mapping in the MWPC case, this is a rather "luxury" way of image data storage, but it pays off in performance. In case of 16k processors one can load into it $s = 800$ lines of a slice with $n = 20$ pixel width (or a square patch of $128 * 128$). It should be remembered that ASP is a STRING, thus pixel neighbours inside a given line are also neighbours in the ASP but the column-neighbour of pixel $[X = n; Y = k]$ is the pixel $[X = 1; Y = k + 1]$.

For each "black-pixel" bit#1 is set to 1 in the associative memory. Counting the number of "black-pixels" inside a given θ corridor with intercept parameter b_i is reduced to a series of shift and add operations in the following way. One selects a "counter processor" in each corridor which is the last processor on the string belonging to corridor i. This C_i counters are preflagged before execution and bits $17, 18, 19$ are cleared. Then one repeats the following cycle till the very first corridor point reaches the position where originally the content of the last corridor pixel was, which contains counter C_i:

a) If bit#1 is set in the preflagged counter processor: $C_i = C_i + 1$;
b) Shift one-by-one steps to right of all bit#1 until the next point in the corridor reaches the original position of the last (the so called "storage-cell") point in the corridor. The number of these steps depends only on θ and it is a priori known in the ASP controller.

By this way one fills in the (b, θ) plot for the whole column simultaneously as it shown in Fig. 22. In this sense one can regard the SAH algorithm as "parallelism of the first power".

Using the associative character of ASP one can create "parallelism of the second power", too. Exactly this is done by the All Angle Simultaneous Hough-transform (A A S H) algorithm. "All angle" means that one requires only 2 passes: the first for tracks between angles -45° and $+45^\circ$ (y-slice scan) and the second for tracks between $+45^\circ$ and $+135^\circ$ (x-slice scan). The number of θ_i values is determined by the number of columns, n in the processed patch, because it is not worth to have different corridors for tracks which are starting from the same point of the first column and whose separation is less than 1 pixel in the last column.

Fig. 22. Single angle Hough-transform

Restricting the angular resolution to this self defining limit the number of necessary "storage cells" is twice as many as the number of pixels per patch, thus assigning two counters to each processor corresponding to positive and negative θ corridors, respectively one gets an ideal match between the size of the (d, θ) "lego-plot" and the iconic image size. In case of the CERN NA35 experiment on the streamer chamber pictures the choice of $n = 20$ gives about 2.3° resolution. Fig. 23 shows four such consecutive slices creating a super-slice which improves the angular resolution proportionally [6]. The tracks which are reconstructed as peaks form the Hough-transform matrix are superimposed. It is remarkable how well this procedure works in this very dense picture fragment.

Fig. 23. All angle Hough-transform

The AASH routine is both efficient and very fast. The processing speed for a slice of size $s * n$ is independent of s, and the whole calculation requires only $n * n$ right-shifts and twice as many additions of $log_2(n)$ bits. In ASP the setting $s = 800$ and $n = 20$ requires $N_{slice} = 4400$ computational steps per slice, which corresponds to a processing time $T_{slice} = 440\mu s$ assuming 100 ns cycle time. For the whole 3 Megapixel image it would take 180 millisecond. As the Hough-transform is done independently for single slices a "parallelism of the third power" is also possible. If the length of the processor string is large enough to load all slices at the same time, then the simultaneous slice processing could bring down the processing time required for the whole image to the level of T_{slice}. Of course, this time takes only into account the table filling of the Hough-transform. The peak finding algorithm requires comparable time slot additionally.

3.4 Track-length method

When the particle is passing through any material then it is losing energy. If the material is dense and/or long enough then it will stop. This range is rather well defined and the track-length is well correlated with the original energy. Relying on the track-length measurement a tracking device can behave like an energy measuring calorimeter. Thus one can perform tracking and calorimetry simultaneously. In order to avoid to go too deeply in technical details it is worth to formulate this problem in a more common-day scenario.

Let us assume there are two lakes A and B, connected to each other by a channel. The channel is crossed by a bridge where the anglers are sitting. In summer all the fishes are living in lake A in complete harmony, but in autumn they migrate to lake B. During this migration it is necessary to make a selection according to the fish length because in lake B the big fishes eat the small ones (and any way the big fishes have the ideal size for human consumption). Net is not good, because the fishes are differing only in length but not in lateral size. Anglers on the bridge should catch only fishes which are longer than a minimal size **independently of the swimming direction** . In case of human actors using the automatic pattern recognition built-in the eye this seems to be a minimal requirement, but it is not so trivial if one tries to realize it in a fast on-line system.

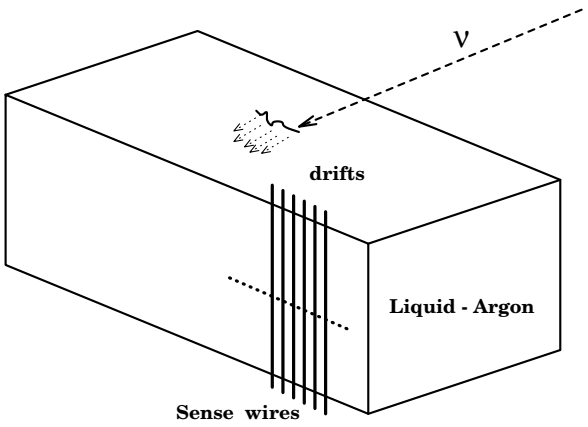

Fig. 24. ICARUS detector

In the analog particle physics experiment (ICARUS) the water corresponds to liquid Argon and the fishes are electron tracks generated by solar neutrino collisions (Fig. 24.), which are drifting toward the detector plane representing the bridge with the anglers. These electrons should have some minimal energy in order to be able to distinguish them from the background noise caused mainly by ^{42}Ar radioactive β^- decays of smaller energy.

In the planned detector the number of sense wires will be about 20 000 with 2 mm pitch. The signal will be read out every 1 μs (million times per second). This produces $2 * 10^{10}$ bit/sec information (**20 Gigabit/sec**!!). Of course, in most of the cases it is empty, but one needs a fast on-line processor which is able to decide promptly when there is something interesting.

The aim of the proposed algorithm is to provide in every μs a decision on that whether between the present moment and 10 μs before was there a track which is at least 10 pixels long. That is a sliding 10 μs test is performed in every microsecond.

In this case the "iconic" representation is understood only in "allegoric" sense, because the processors will contain at a given moment only one time slice of the image drifting through the detector, as it shown in Fig. 25. where the horizontal axis corresponds to the usual processor string, but the vertical axis is the time axis, denoting the image lines reaching the sensors 1,2,...,n,... microsec later.

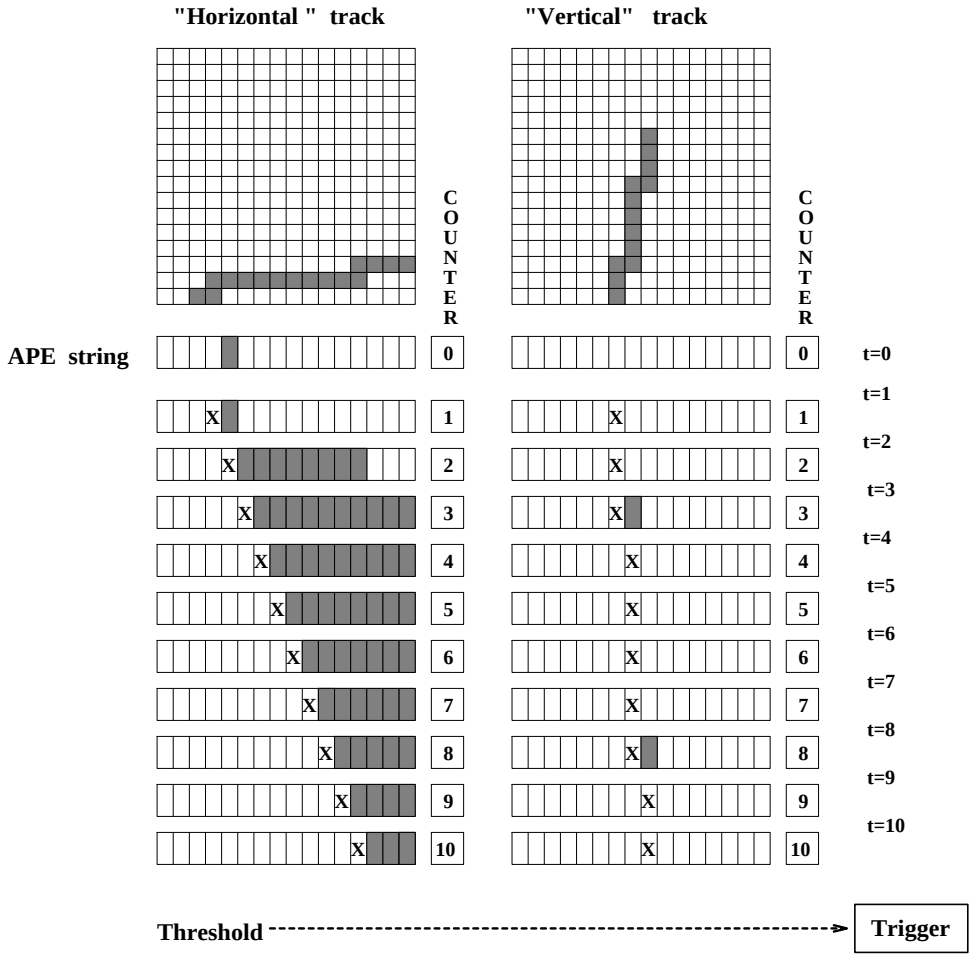

Fig. 25. Horizontal and vertical tracking

The essence of the algorithm that it makes a tricky "convolution" by mixing the previous state of the processor string and the new incoming image line. The basic flow-chart is shown in Fig. 26. It looks very simple because all the complications are hidden in the fact that this is parallelly executed by 20 000 processors simultaneously.

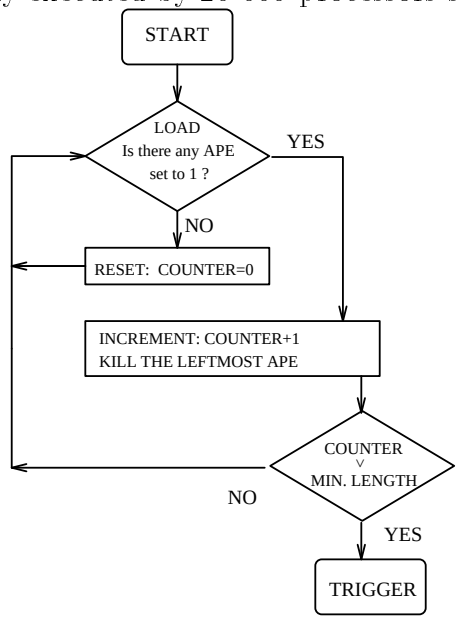

Fig. 26.

In every microsec there is a LOAD-ing which means that the image-bit in the memory

is set to 1 if it was 0, but the incoming 0 doesn't clears the bit if it was set to 1 previously. Fig. 25 shows two typical tracks: the first is almost parallel to the sensor plane and most hits are arriving within a short time range; the second track is almost vertical therefore the hits are arriving one by one during 10 microsec. The real counting in preflagged processors is performed only for the leftmost hit in the loaded string. Only the counted hit is set to zero leaving alive the "old" uncounted pixels, thus storing the information from horizontal tracks for later counting.

Segmentation of 16 processor length is introduced along the string thus only the counts from a small neighbourhood are taken into account. By "double" counting the algorithm is working equally well for tracks crossing the segment boundary.

This storing, erasing and segmentation tactics assures that the track-length calculation is not sensitive for the track direction. In case of a physically well applicable algorithm one should provide some robustness against detector deficiencies. The above algorithm can tolerate "fat" tracks and it can survive small inefficiency gap. It is not, however, a real tracking algorithm because no check of smooth continuity is made. One can invent a number of very artificial pixel distributions which would pass the test requested by this routine, but these are so artificial that doesn't have practical significance.

3.5 Cellular automata

Though it is only indirectly related to tracking and HEP, it is worth to mention how good can be the ASP machine for cellular automata applications.

The cellular automata (CA) are automata networks, invented by J. Neumann to model discrete dynamical systems. They can describe the complex, collective behaviour and time evolution of systems of many degrees of freedom in physics, chemistry, biology or computer science. Unlike ordinary differential equation description they can encounter "strange" self-organizing systems. The CA, as shown by Wolfram, can be classified according to their evolution to:

class 1. simple homogeneous state;

class 2. periodic state;

class 3. chaotic state;

class 4. towards states with complex localized or propagating structures.

The fine grain of parallelism, neighbour connectivity and the simple logical manipulation capability of ASP makes it an ideal area for cellular automata simulation. By assigning one processor/cell, maximal parallelization can be achieved for a CA with synchronous rule.

To demonstrate this we programmed two recently investigated 1-dimensional CA named Rule-18 and Rule-54. These are two-valued, range-one deterministic automata, still showing complex collective behaviour. They belong to class 3.

RULE-18:

$$f_{18} = 1 if (x_1, x_2, x_3) = (1,0,0) or (0,0,1); \qquad 0 \quad otherwise$$

RULE-54:

$$f_{54} = 1 if (x_1, x_2, x_3) = (1,0,0), (0,0,1), (0,1,0) or (1,0,1); \qquad 0 \quad otherwise$$

Where x_2 is the updated processor and x_1, x_3 are the left and right neighbours, respectively. For the Rule-18 defects of the pattern show deterministic diffusion behaviour while in Rule-54 different particle-like structures appear, possessing complex interactions, as it is shown in Fig. 27.

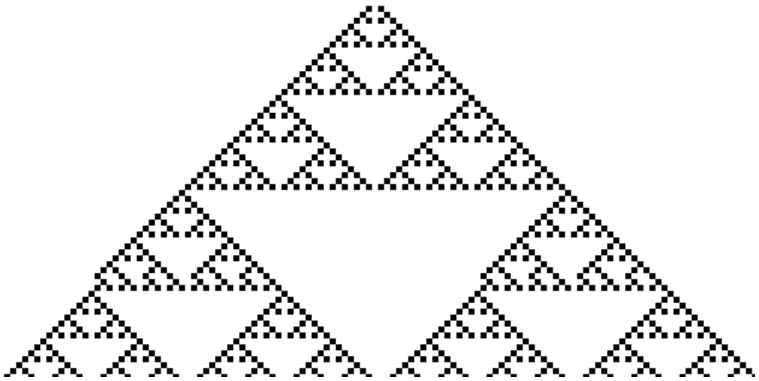

Fig. 27.

Using special parallel random generators in ASP one can study probabilistic CA too, where the value is set only by probability P. It is easy to show that below about $P = 0.8$ the chaotic behaviour of Rule-18 disappears and the pattern dies out.

4. THE ASTRA MACHINE

Four ASTRA machines were produced within the CERN-MPPC project lead by F. Rohrbach in Brunel, in Saclay, in Orsay and in CERN. These machines are development systems therefore they look a bit more complicated than it is really required.

4.1 Hardware installation

The present systems are running with 2 ASPA cards providing 4096 processor chains. Fig. 28 shows the logic scheme of the machine. By adding more ASPA processor cards one can extend the system up to 16k processors.

The SUN workstation (used as human interface to the machine) which is called the High-Level ASP Controller (HAC) drives the FIC (a commercial 68030 VME CPU board) called the Intermediate-Level ASP Controller (IAC). The ASP machine itself is composed of one Low-Level ASP Controller (LAC) card and a maximum of eight ASP Array (ASPA) boards. On the LAC board the microcode memory stores the command sequence to be executed by the ASP string. The ASPA cards comprise four ASP channels with eight VASP-64 chips per channel, each chip contains 64 Associative Processing Elements (APEs).

The backbone of the ASP machine is a proprietary bus, the ASP Global bus (AGbus). The communication between the LAC and the ASP array is done by this AGbus on the P3 connector of the extended VME card.

The scalar data are transmitted on 32 lines in the form of a 32-bit binary word or in ternary mode in case of bytes or bits. The activity bits use 12 lines and are always transmitted in ternary mode.

The instructions from the LAC are transferred over AGbus in a compressed form (20 bits). They are expanded inside the ASP boards. The AGbus also carries synchronization signals and status (e.g. match). Four daisy-chains are used between boards in the AGbus backplane to implement the inter-APE network.

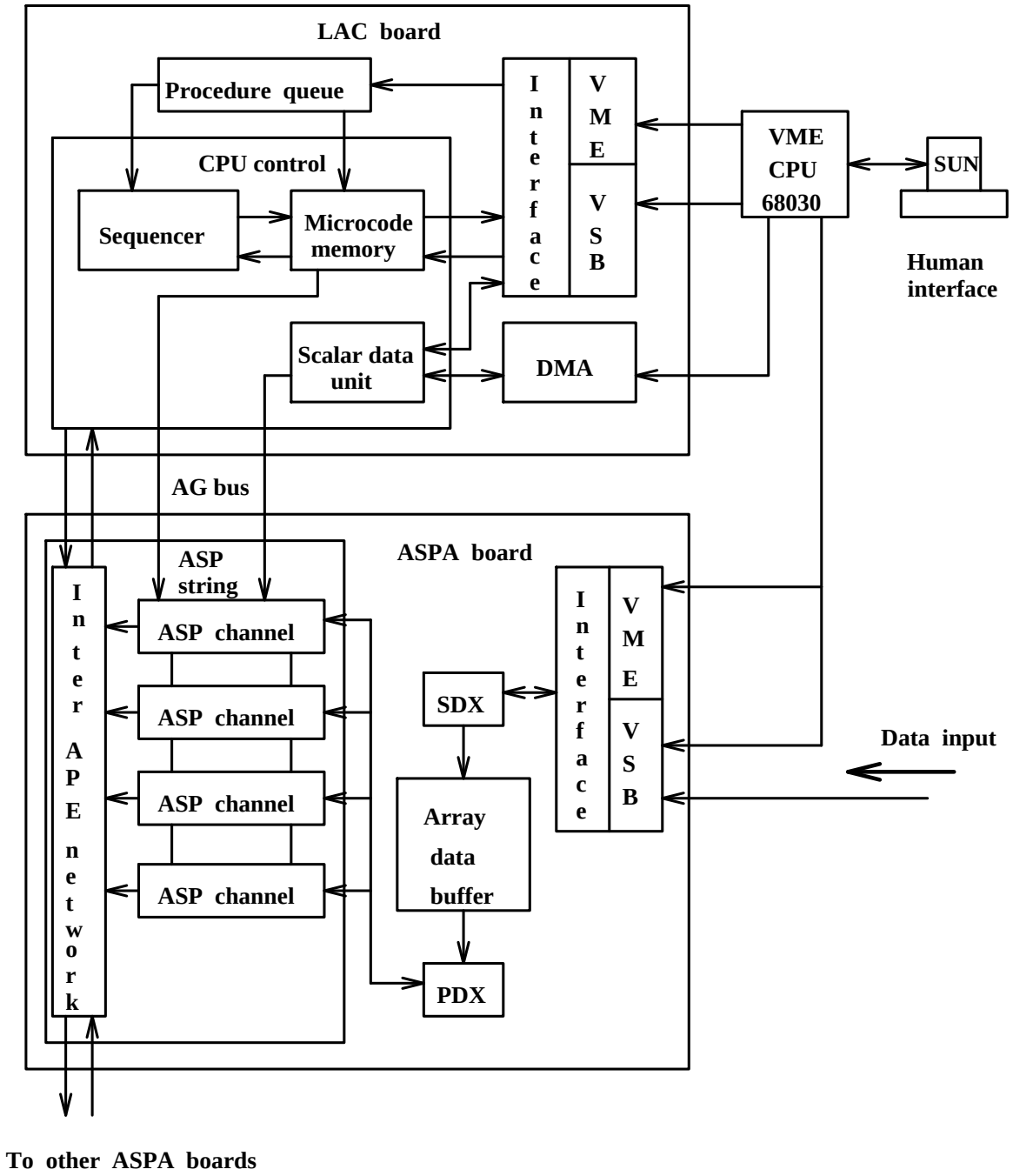

Fig. 28.

4.2 Associative Processing Element (APE)

The basic building block of the system, the associative processing element (APE) is shown in Fig. 29. which one can call also as "intelligent memory cell" at the same time emphasizing its simplicity and expressing the intermediate evolutionary step toward real intelligence in a bit similar way as the monkeys are regarded somewhere between human and animal worlds.

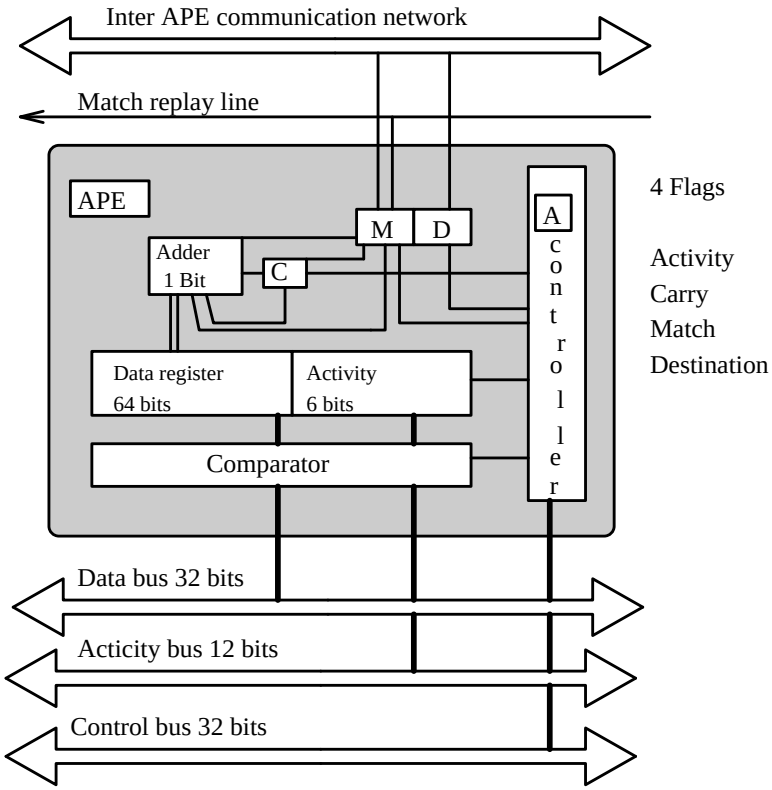

Fig. 29.

The associative memory part comprises the Data Register (64 bits) and Activity bits as 6 independent bit-registers. Both the Data and Activity bits can be content addressed by help of the COMPARATOR through the overall Data and Activity buses. The Data Register can be loaded from the Data bus, but Activity bits can be set only by internal APE operations. The state of APE is determined by four flags:

Activation flag (don't mix with activity bits);
Carry flag (stores the overflow in single-bit addition);

Match flag (Tagging register#1 = Tr1);

Destination flag (Tagging register#2 = Tr2).

As a result of "associative (tagging) operations" when subset selection is performed a universal MATCH-REPLY is worked out which represents the *OR* of all the M-flags along the string.

The inter-APE communication network is able to transmit 1-bit information to left or right directions changing the setting of M,D and A flags.

By software setting (i.e. by a single instruction) it is possible to redefine in any moment the segment length. This provides a very flexible facility for programming because e.g. in one moment of the program it can be regarded as a SINGLE STRING with 4096 APEs and in the other moment as more than 500 INDEPENDENT 8-processors STRINGs.

4.3 Basic programming cycle

The big invention in ASP programming is that the operations are generally not executed by the subset of those APEs which provided the positive MATCH-REPLY, though this case is not excluded either. One should imagine what a powerful variable tool is created by separating the **matching step** from the **activation step**. This is the reason why are separate M,D tagging flags and the A activation flag. The large variety of ACTIVATION MODES is illustrated in Fig. 30 by some examples.

All ACTIVATION MODE enumerations except *a* and *am* can be grouped into pairs. Each enumeration in a pair has the same functionality but acts in opposite directions. All

those enumerations that can be grouped can contain either the letter "l" or the letter "r", corresponding to the leftward and rightward directions in which they act. The following is a list of the enumerations and their corresponding functionality:

a activate all APEs **tagged** in Tr1
am activate all APEs **not tagged** in Tr1

anl,anr activate the **neighbours** in the specified direction to those APEs tagged in Tr1

afl,afr activate the **first** APE tagged in Tr1 from the specified direction

all,alr for every APE tagged in Tr1, activate the **subsequent** APE in the specified direction which is tagged in Tr2 (selective alternate tagging for Tr2 used for e.g. n term summing in $log_2(n)$ steps)

ael,aer activate **every** APE in the specified direction to those tagged in Tr1

aelf,aerf activate **every** APE in the specified direction to those tagged in Tr1 **including** those tagged in Tr1

asl,asr activate all **substrings** of APEs in the specified direction up to the subsequent processing element tagged in Tr2

aslf,asrf activate all **substrings** of APEs in the specified direction and **including** those tagged in Tr1 up to the subsequent processing element tagged in Tr2

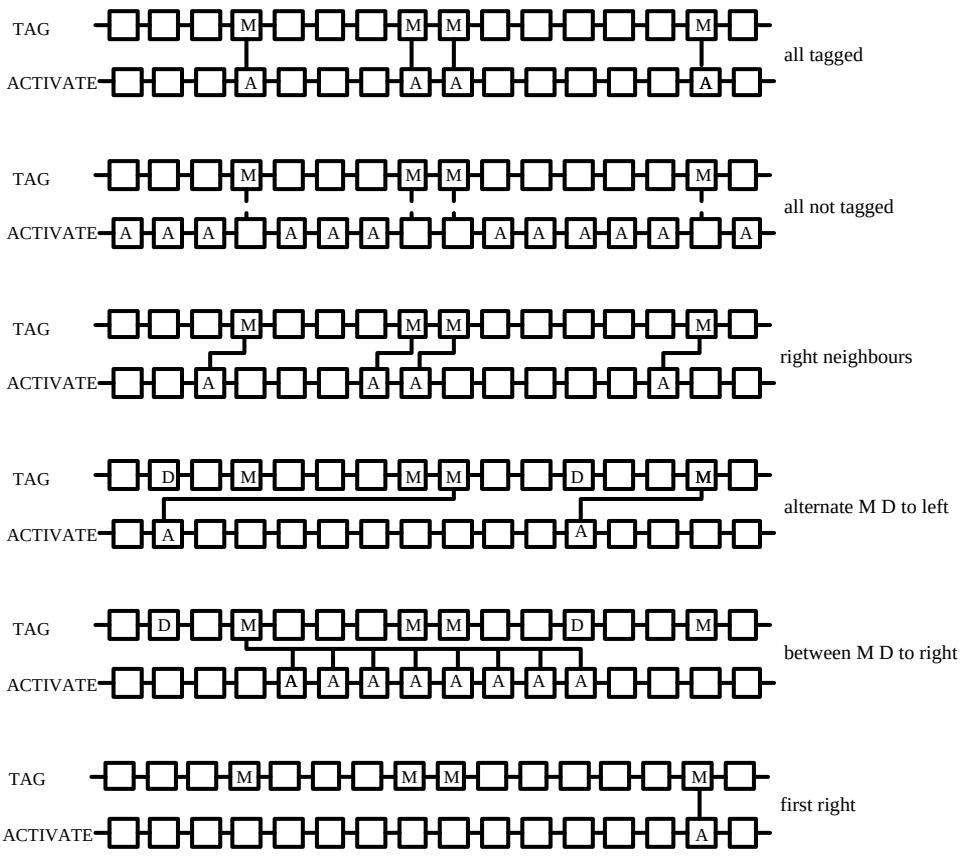

Fig. 30.

The use of two types of tagging flags ($M = Tr1$, $D = Tr2$) indicates that even the tagging procedure is not as simple as it looks for the first glance. One has the possibility to choose between three TAGMODES:

a) In tr1 mode all APEs with successful match will be tagged in Tr1;
b) In tr12 mode all matching APEs will be tagged in Tr1 **AND** Tr2;

c) In trAlt mode the matching APEs will be tagged **alternatively** starting by Tr2.

In summary the basic programming cycle in ASP consists of the following steps: tagging, activation and execution. Of course, this is only the standard procedure, in real applications there are more variants which are not discussed here because they are interested only for the specialists.

4.4 Operation types

Due to the fact that the data are stored in a number of different positions one can perform a large variety of operations in various scenarios. Here we mention some of them only for illustration.

As it was mentioned above there is a very convenient data structure the "serial field" grouping together any number of subsequent data bits inside the Data Register of each APE. This entity corresponds to the commonly used "integer" data type in other programming languages with the dynamical length and position definition which can be redefined in any moment. At a given time three disjunct serial fields $(sf1, sf2, sf3)$ can exist simultaneously their partial or complete redefinition is done without real loss of computing time because one needs to set only some pointers in the LAC.

One basic type is the **scalar-vector** operation when each selected APE has its own source information in one of its serial fields which is modified by a common number broadcasted on the Data bus (Fig. 31.a).

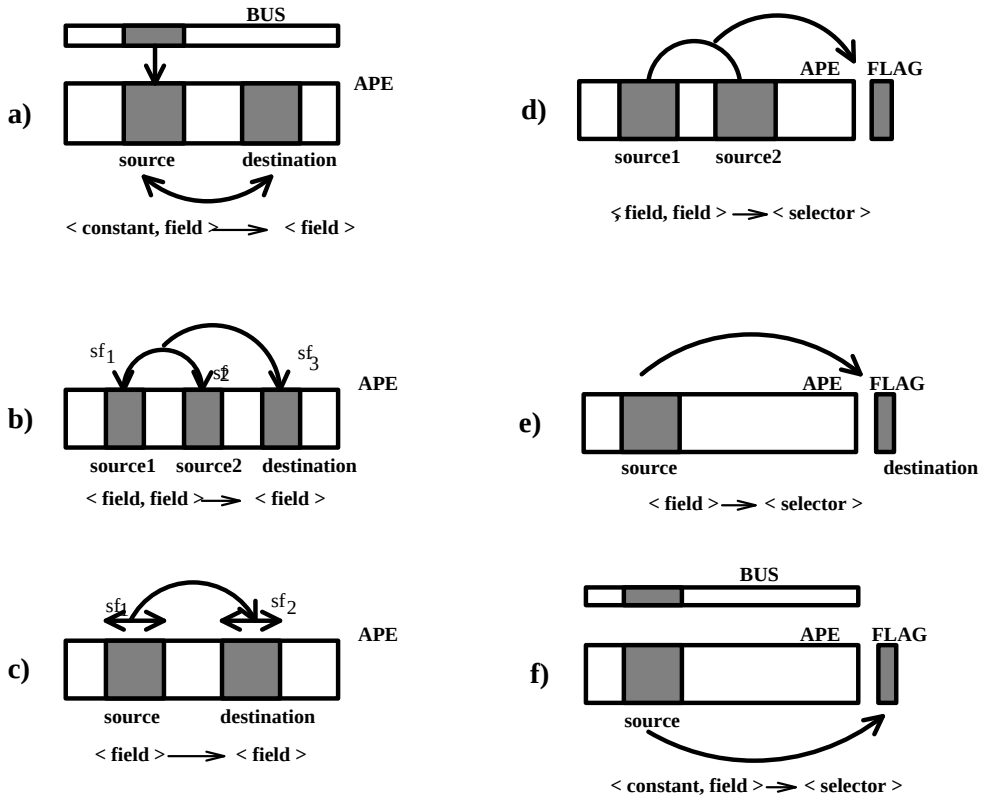

Fig. 31.

The other basic type is the **vector-vector** operation, when in each APE one can calculate from two source fields according to some rule (add, multiply, logical AND etc.) a third one which is put into the destination field (Fig. 31.b). It can happen that only one source is required like in case of copy (Fig. 31.c). It is also possible that one is interested only in a single-bit MATCH-REPLY (e.g. mark the case when $sf1 = sf2$ (Fig. 31.d);

mark APEs which have greater number in sf1 than the one sent by the Data bus (Fig. 31.f); mark the APE with maximum (Fig. 31.e).

4.5 Programming environment

The target machine, as described in the previous sections, consists of several hardware modules. An ASTRA application requests the programmer to support a software main module for each hardware level of the machine: he/she has to write the HAC, IAC and LAC modules defining the interfaces between each module. A number of facilities are available in order to write this kind of three-level application.

The main idea is to program each layer using services provided by the lower layer. That is, the HAC part will use the IAC procedures in remote to drive execution on the IAC, and the IAC part will use the LAC procedures in remote to drive execution on the LAC. The system so defined uses a Cross Procedure Call (or Remote Procedure Call) mechanism for control and data communication between each hardware level. The procedures defined in one module and called in one module of a different level are called cross-exported procedures. The way the cross procedure calls is completely transparent to the application programmer.

The language used for writing an ASTRA application is Modula-2 with some restriction for each level. Since the target machine consists of three different hardware modules, three different commercial compilers are used to generate the corresponding target executable code. The three compilers are:

gpm: Garden Points Modula-2 for Sun SPARCstation architecture;
ace: The ACE Cross Modula-2 Compiler for MOTOROLA 68030 architecture;

lamc: The ASPEX Microsystem Ltd. Cross Compiler for Low-Level ASP Controller.

In order to simplify the task of the programmer, a multi-level compiler generator is provided. The programmer has to write the definition and implementation modules for each level. Those modules will be compiled using a program called "aspc" which drives the execution of the right compiler according to the target level that the module represents. Furthermore, the "aspc" compiler will generate the necessary code for implementing the cross-exported procedures.

The compiled modules are linked to form a single application program ready to be executed on the three-level hardware components of the machine. The linker program is called "aspl" which links the modules taking into account the three different hardware targets.

Before the execution of the user code, the system checks the status of the ASTRA machine. In particular, it initializes the different hardware components and down-loads the necessary codes to the IAC and LAC levels.

5. PERSPECTIVES

Associative processing involves a particularly flexible and naturally parallel form of iconic representation and manipulation of structured data (viz. sets, arrays, tables, trees and graphs) processing with potential benefits in simplicity of expression, storage capacity, and speed of execution over a wide range of non-numerical and numerical information processing applications.

The potential offered by the ASP architecture stems from the recognition that dynamically reconfigurable heterogeneous low-MIMD/high-SIMD architectures could offer the most cost effective solution for second-generation massively parallel computers (MCPs).

Ideally, each processing node of such a heterogeneous MPC would comprise a medium-grain microprocessor which would be tightly coupled to a fine-grain SIMD processor. Each node could execute a different task, thereby achieving the maximum MIMD processing

power. However, in operation, virtual nodes comprising dynamically reconfigurable clusters of physical nodes (each executing the same task) would be formed to match the natural parallelism of the application.

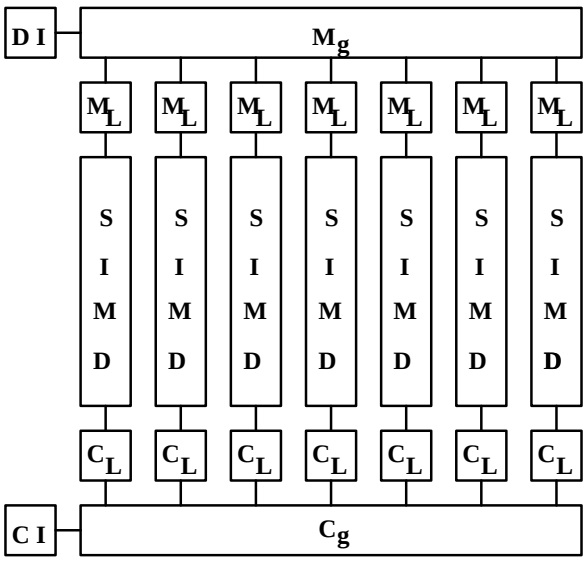

Fig. 32.

A simplified schematic of a dynamically reconfigurable low-MIMD/high-SIMD second-generation MPC that could be attached to an appropriate data source (via data interface DI) and a host computer (via control interface CI) is shown in Fig. 32. In operation, the global controller C_g could configure this heterogeneous MPC, either as purely SIMD machine or as many low-MIMD/high-SIMD variants, by allocating local controllers C_l, to the same or different tasks. Communication between the SIMD processors could be achieved via the shared memory M_g.

5.1 ASP modular systems

According to application requirement, an appropriate combination of ASP modules would be attached to the control bus and the Data Communication Network as indicated in Fig. 33. This could be constructed so as to implement any general purpose network topology (e.g. cross-bar, mesh/torus or binary n-cube) or application specific topology (e.g. shuffle, exchange or butterfly) to enable data transfer between pairs of selected ASP modules.

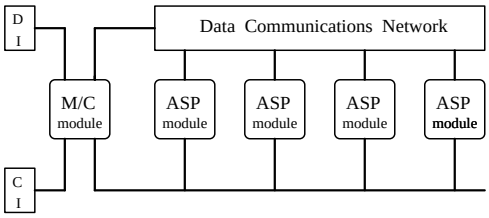

Fig. 33.

The ASP module offering the highest performance parallel processing power with largest data input-output bandwidth corresponds to the configuration indicated in Fig. 34. This has a single CI and multiple DIs.

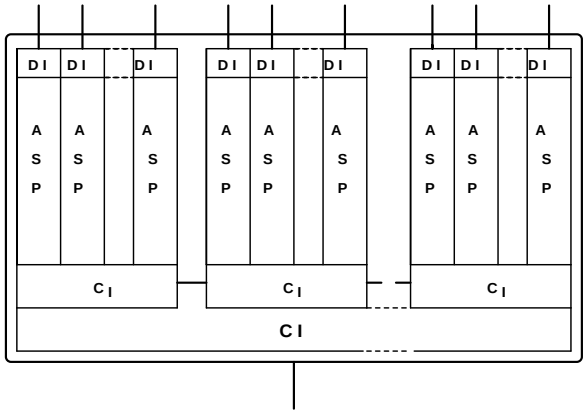

Fig. 34.

The basic structure of the architecture, the ASP substring is a programmable, homogeneous and fault-tolerant fine-grain SIMD machine incorporating a string of identical associative processing elements (APEs), a reconfigurable inter-processor network and a Vector Data Buffer (VDB) for overlapped data input-output, as indicated in Fig. 35.

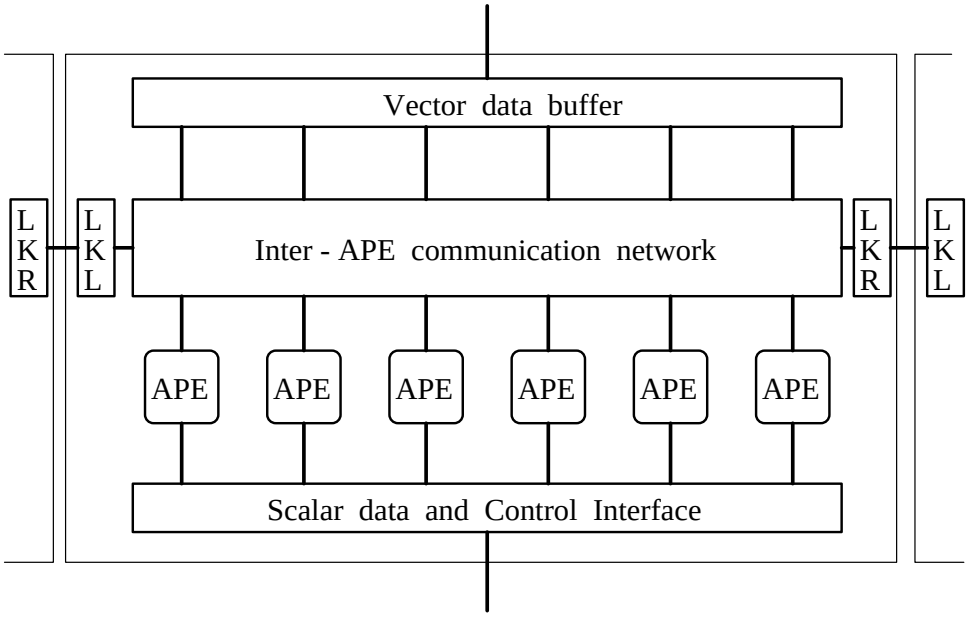

Fig. 35.

The network supports simple modular extension via the Link Left (LKL) and the Link Right (LKR) ports enabling the user to simply extend the string length. The network supports circuit-switched asynchronous communication, as well as more conventional synchronous bidirectional multibit communication. At an abstract level, a circularly-linked ASP substring can be considered as a hierarchical chordal-ring with the chords bypassing APE blocks (and groups of APE blocks), thereby accelerating inter-APE communications signals.

The upper levels of the input-output hierarchy consist of paging information between external global memory (M_g) store and the local memories (M_l) as indicated in Fig. 32. At the lowest level of the data buffering hierarchy, the Vector Data Buffer (Fig. 35.) supports a dual-port exchange of vector data with the APE data registers (primary data exchange PDX) and with the local memories (M_l)(secondary data exchange SDX).

The bit-serial (APE-parallel) PDX is a very high bandwidth exchange. The lower band-width bit-parallel (APE-sequential) SDX provides a slower data exchange between

the VDB and the M_l, but may be fully overlapped with parallel processing and so does not necessarily present a sequential overhead.

The ASP concept has been specifically developed to exploit the opportunities presented by the latest advances in the VLSI-to-WSI technological trend, high-density system assembly techniques and state-of-the-art packaging technologies.

5.2 VLSI development program

The key processing component of an ASP is the ASP substring, shown in Fig. 35. When implemented as a VLSI memory structure, this results in a very dense and regular implementation. Following extensive research prototyping, a commercial 64-APE VLSI ASP chip has been developed by Aspex Microsystems for fabrication ($2\mu m$ CMOS with 2-layer metal) by ES2 (France). Incorporating 64-bit data registers and 6 activity bits, the device is the first example of a family of devices intended to benefit from rapid scaling down to near $1\mu m$ feature sizes.

Current development systems support 16k APEs with a single controller, but facility for simple expansion is readily available in keeping with the overall ASP strategy. Indeed, ongoing research into hybrid technologies points towards multiple ASP devices (either the current generation or next generation 256-APE chips) integrated on a single silicon or ceramic substrate, thereby increasing the single-board substring to at least 16k APEs. Performance values for 16k/64k APE boards, which can be envisaged as coprocessor boards in standard workstations, are indicated in Table 3.

Fixed-point arithmetic	8-bit [Giga-OPS]	12-bit [Giga-OPS]	16-bit [Giga-OPS]
Add/Subtract	36/150	27/110	21/84
S-V Multiply	4.9/20	2.4/9.6	1.5/5.9
V-V Multiply	2.7/11	1.3/5.2	0.8/3.1

Floating-point arithmetic (32-bit IEEE)	
Add/Subtract	0.22/0.90 - 4.5/18 [Giga-OPS]
S-V Multiply	0.86/3.5 [Giga-OPS]
V-V Multiply	0.24/0.97 [Giga-OPS]

Table 3. Performance Values for 16K/64K boards at clock rate = 80 MHz

5.3 WSI associative string processor

The WSI Associative string processor (WASP) represents a challenging and innovative method of implementing ASP modules by realizing ASP substring on a single undiced quadrant of a silicon wafer.

As indicated in Fig. 36., a WASP device is composed from three different modules implementing Data Routers (DRs), ASP substrings, and Control Routers (CRs). The DR and CR blocks incorporate routing to connect ASP substrings rows to a common DI and common CI, respectively.

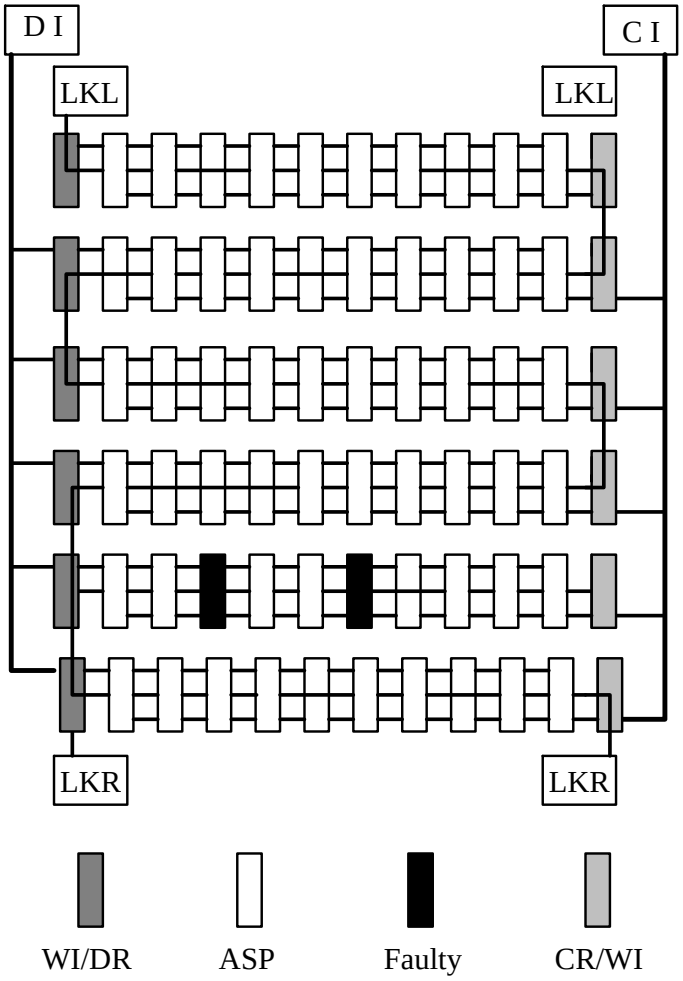

Fig. 36.

Each of the DR, ASP substring, and CR blocks integrates a defect/fault tolerant design; such that APE blocks within the ASP substring blocks, and entire ASP substring rows (and, hence, entire WASP devices) can be bypassed. In practice, WASP device form factors depend on the packaging standard adopted. Of these standards a light-weight SEM-E compatible module is assumed for the following projections. The SEM-E modules are based on a 6.4 in * 5.88 in thermal conduction plate with a substrate, supporting micro-electronic circuitry, attached to each side. Overall thickness of the package enclosure is 0.6 in. One such substrate supports four WASP devices, where a fully developed target WASP device can comprise a 15360 APEs array in a 6 cm * 6 cm chip. Allowing for the expected loss of processing elements through defects, a SEM-E compatible package can therefore support a 65536-APE WSI ASP module with eight (32-bit) data channels. In a longer term, 3D-WASP architectures can provide even more attractive performance (see Fig. 37.).

6. CONCLUSIONS

I hope that the talk and demonstration was convincing enough to prove that the ASP principle and the ASTRA machine works.

This machine is well adapted for bit (and integer number) manipulating ICONIC algorithms which can be very effectively applied in a number of high energy physics experiments.

The other advantage of this architecture that it could provide "unlimitedly" high I/O band-width, which could be also very attractive for high luminosity HEP experiments, but this "straight-forward" possibility was not yet realized in existing systems.

There has been developed very efficient software tools, but more progress is required both on the system - and the application software level.

In summary, on can say that ASP is not the best universal computer, but for a large class of the problems it can provide probably the most optimal solution. In order to fulfill these promises one need cheap commercial components, modules and standard coprocessor cards, which makes them easily available in case of eventual occurrence of demand.

In these sense Zanella's final words from his '92 talk seems to be still valid:

> Massively "Parallel Computing is the technology of the future and always **will** be.... The future is getting closer....."

As a personal remark I should like to add:

> There should be something in the idea of massive parallelism and associativity because there exists at least one rather successful realization along these design principles: the human brain.

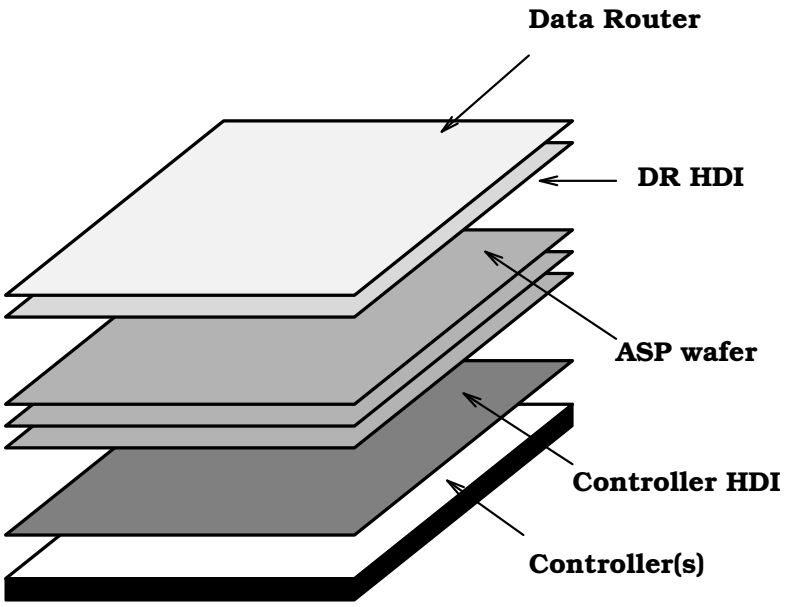

Fig. 37.

REFERENCES

[1] P. Zanella:*Status of Parallel Computing*, Proceedings of 1992 CERN School of Computing, L'Aquila, CERN 93-03, p.7.

[2] D. Parkinson:*SIMD Processor Arrays*, Proceedings of 1991 CERN School of Computing, Ystad, CERN 92-02, p.204.

[3] R. Shepherd:*Next Generation Transputers*, Ystad, Proceedings of 1991 CERN School of Computing, Ystad, CERN 92-02, p.299.

[4] B. Thooris et al.:*ASTRA-2 tutorial*, CERN/ECP, MPPC-94/.. and CERN-ASPA User's Book, CERN/ECP/RA1, MPPC/93-96

[5] R. M. Lea:*ASP, a cost-effective parallel microcomputer*, IEEE Micro, Oct. 1981, 1.

[6] G. Vesztergombi:*"Iconic" tracking algorithms for high energy physics using the TRAX-1 massively parallel processor,* in Proc. Conf. on Computing in High Energy Physics, Oxford, 1989, Comput. Phys. Commun. **57** (1989) 296.