Università di Pisa

The Tree Search Processor for Real-Time Track Finding

Antonio Bardi
Dipartimento di Fisica
Università di Pisa
Mauro Dell'Orso
Dipartimento di Fisica
Università di Pisa
S. Galeotti
Istituto Nazionale di Fisica Nucleare,Pisa
Paola Giannetti
Istituto Nazionale di Fisica Nucleare
Giuseppe Iannaccone
Dipartimento di Ingegneria dell'Informazione: Elettronica, Informatica, Telecomunicazioni,
Università di Pisa
E. Meschi
CERN, Div. PPE,
Svizzera
F. Spinella
INFN di Pisa

A. Bardi, M. Dell'Orso, S. Galeotti, P. Giannetti, G. Iannaccone, E. Meschi, F. Spinella, *The tree search processor for real-time track finding*, 1998 IEEE Nuclear Science Symposium Conference Record.1998 IEEE Nuclear Science Symposium and Medical Imaging Conference (Cat. No.98CH36255). IEEE, Piscataway, NJ, USA, 1998, vol.2, p.969-973.

The Tree Search Processor for Real-Time Track Finding

A. Bardi[2], M. Dell'Orso[1], S. Galeotti[2], P. Giannetti[2], G. Iannaccone[3] E. Meschi[4], F. Spinella[2]
[1]Dipartimento di Fisica, Università di Pisa, Piazza Torricelli 2, 56100 Pisa, Italy
[2]INFN Pisa, Via Livornese 1291, 56010 S. Piero A Grado (PI), Italy
[3]Dipartimento di Ingegneria dell'Informazione, Università di Pisa,Via Diotisalvi 2, 56126 Pisa, Italy
[4]CERN/Div. PPE, CH-1211 Geneva, Switzerland

Abstract

We propose a dedicated parallel and pipelined hardware to implement a binary search strategy, for real-time track finding in high-energy physics, based on a large bank of pre-calculated combinations of trajectory points. The bank is organized into a hierarchical structure whose levels correspond to the pipeline stages. Many state machines compare independent bank sections to the event for track identification. High density commercial RAMs store the bank; the machines are easily packed into FPGA devices.

I. INTRODUCTION

The tree search algorithm [1] is an efficient solution to the pattern recognition problem for real-time tracking of very high multiplicity events in high-energy physics. It can be efficiently executed by standard CPUs or implemented in a parallel and pipelined dedicated hardware, the *Tree Search Processor* (TSP). A first TSP prototype has been built in the past [2], but its old architecture does not implement the parallelism proposed here.

TSP has been proposed as a second stage of a 2-level system [3], where the first stage is implemented by AM [4]. AM is an associative memory which performs tracking at the detector readout rate, recognizing low-resolution tracks called fat roads. For each event, TSP receives fat roads and high-resolution hits and it refines the pattern recognition at the AM output rate.

II. THE ALGORITHM

The algorithm [1] is based on the idea of a large bank of precalculated patterns. It solves the track finding problem in a detector section, called *fat road*, consisting of a number of *layers*, each layer being segmented into a number of *bins*. When charged particles cross the detector they hit one bin per layer. The problem of track finding is reduced to a search in the bank for hit patterns matching the event. The pattern bank can be organized into a special structure to speed up the pattern matching process.

The basic idea is to follow a successive approximation strategy and apply the pattern-matching algorithm to the same event seen with increasing spatial resolution. Lower spatial resolution is simulated by logically ORing adjacent bins.

Figure 1a shows how a single track, crossing four parallel layers, is seen when each road plane is considered as being only two bins. In this case, the total number of patterns compatible with a straight line is eight. Pattern number 3 is the one that matches. Since we have one track candidate at this level of spatial resolution, we now double the number of distinguishable bins in each plane and proceed to match the four patterns shown in figure 1b. Pattern number 3 in figure 1a is called a *parent pattern* and it is said to generate the four sub-patterns of figure 1b. Since we still have one track candidate we go on halving the bin size. This process is iterated until we either reach the TSP final resolution (success) or we are left with no track candidate (failure).

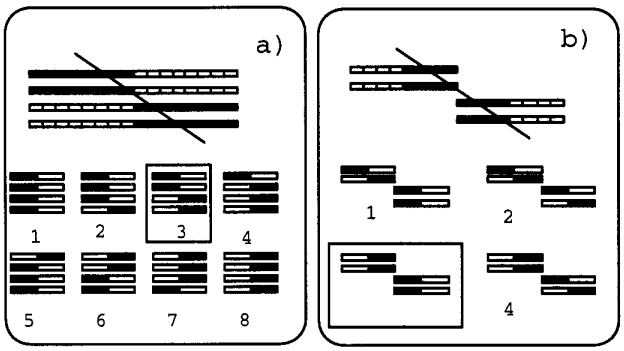

Figure 1: (a) Eight patterns are compatible with a straight line when each layer is divided in two bins. Pattern number 3 is the one that matches. A straight line is used for sake of simplicity; other line shapes would require different patterns. (b) We double the number of distinguishable bins in each plane and proceed to test the four sub-patterns of the matching pattern.

The pattern bank can thus be arranged in a tree structure (figure 2a): increasing depth corresponds to increasing spatial resolution. The tree root corresponds to the incoming fat road at lowest resolution. Each node represents one pattern and is linked to the sub-patterns (*pattern block*) generated when the spatial resolution is improved by a factor two. We scan the pattern block (*block test*) and every pattern matching the event is a track candidate that enables the search at the next level. A refined track candidate (*thin road*) is found whenever the tree bottom is reached. If TSP reaches the detector intrinsic resolution, the pattern recognition is complete; otherwise, the few remaining hit ambiguities are resolved by a further processor, which fits the residual combinations.

This tree-search is obviously much faster than a purely sequential search. The average number of patterns one has to examine to find a single track [1] is proportional to the total number of levels in the tree and to the average number of patterns in a pattern block.

If more than one track is inside the fat road, the number of visited nodes during the tree search increases more than

0-7803-5021-9/99/$10.00 © 1999 IEEE.

linearly [1], since many fake matches can occur at low resolution.

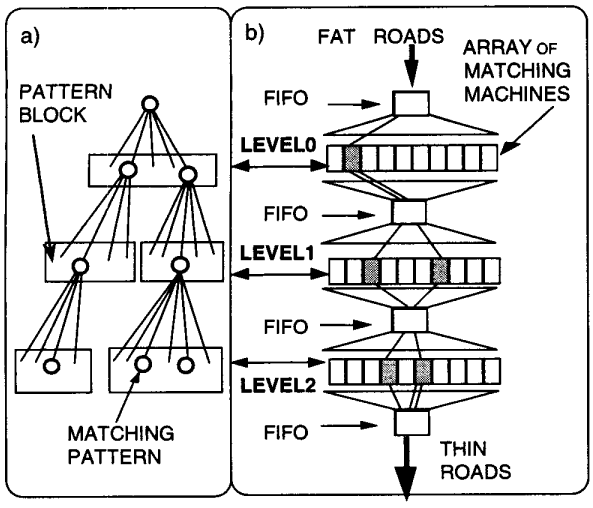

Figure 2: (a) Hierarchical pattern organization in a tree structure. (b) Parallel architecture of machines distributed in pipeline stages corresponding to the tree levels.

III. THE TSP ARCHITECTURE

The tree search algorithm can be easily implemented on a parallel architecture, because different patterns can be compared independently to the event data. Moreover, the tree search can be pipelined, since all tests of one level must be executed after the tests of the previous level. It is possible to reach a high degree of parallelism if we assign several simple machines to the parallel execution of the block tests of each level in the tree. Figure 2b shows an architecture where the machines are distributed in pipeline stages separated by derandomizing FIFO memories.

At level-0, each incoming fat road requires the execution of a single block test assigned to a single matching machine (represented by a gray rectangle in figure 2b). The block test at level-0 generates a variable number of *matches*, corresponding to as many block tests to be executed at level-1. For each level-0 match, the information necessary to process the corresponding pattern block at level-1 is put in the FIFO memory between the two levels. From there, the information is fetched and assigned to a level-1 machine. The process, started by an input fat road, is repeated at every level until exhaustion of all matching nodes. Each road, after level-0, fragments into independent pieces through different tree branches.

The pipeline of figure 2b can handle many roads at the same time. When a new fat road arrives to the TSP, its full resolution hits are distributed in parallel to all the matching machines in the system. The machines have buffers to store hits from different fat roads. This allows any machine to start working on any road at any time.

Fragments belonging to different roads can mix and pass each other along the tree. On the contrary, different events flow in the pipeline keeping the order they entered the TSP. A certain pipeline stage does not process any road fragment belonging to a new event, while fragments of a previous event are still under examination in the same stage.

For an efficient use of the computing power, the number of machines on each stage and the number of buffers per machine must be optimized to match the rates of consecutive stages. To this purpose, the number of machines on a level must be proportional to the average number of matches occurring at that level. Since the number of matches at each level does not grow too rapidly with track multiplicity [1], the number of machines can be reasonably small.

A. Pipeline Stage

Descending one level in the tree structure, the effective spatial resolution increases by a factor two. This means that each bin is split in two halves (figure 1). Starting from a given node, each branch is identified by one bit per layer, which distinguishes the two half-bins: "zero" indicates the right section and "one" indicates the left section. This set of bits is referred to as the *branch*.

When examining one specific sub-pattern during a block test, a branch is called the *current branch*. A node, with its hanging block, is identified by all the branches linking it to the root. They are called the *parent branches*.

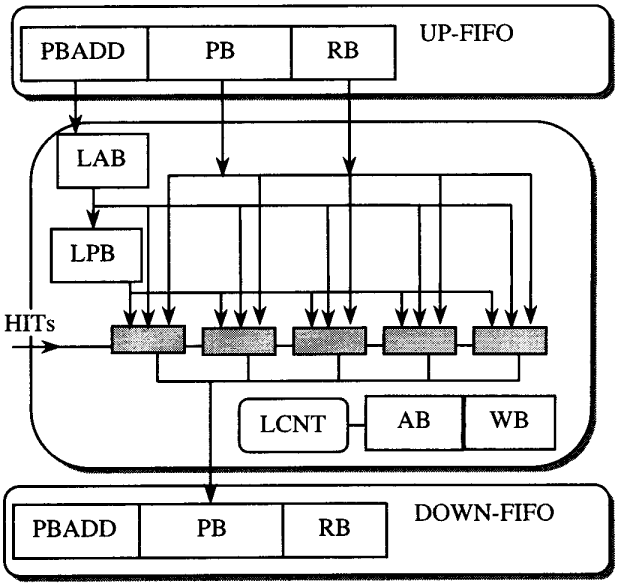

Figure 3: Pipeline stage between the UP and the DOWN FIFOs. The matching machines are represented by gray boxes.

Figure 3 shows the internal organization of one pipeline stage in the TSP. Each machine, represented by a gray rectangle, is equipped with the match logic described in subsection D. Information on the block test to be executed is stored in the up-FIFO: one word per block test. A Level CoNTroller (LCNT) pops the FIFO and assigns the block test to a free machine, which tests one pattern per clock cycle. Hence, the machines are efficiently fed only if their number does not exceed the average number of patterns in a block. Beyond this limit, the FIFO bandwidth is saturated and

machine dead time can be avoided only by improving the communication between consecutive levels.

The information stored in each FIFO word consists of three parts:

1. RB field.
   Each matching machine uses private RAMs to implement several *Road Buffers* (RB). When the TSP receives a fat road with full resolution hits, the hits are sent in parallel and stored into one RB for each matching machine, with the resolution appropriate to the level the machine belongs to. Multiple buffers allow simultaneous memory of multiple roads. For a given input road, the buffer ID is unique and the same for all the machines in the system. During a block test, a machine compares a number of patterns to the hits stored in the buffer identified by the RB bit field.
2. PB field.
   The *Parent Branch* (PB) bit field describes the parent branches from the tree root to the current level. At tree level n, it consists of n bits per detector layer. The parent branches are common to all nodes in the pattern block under test. Patterns in the same block differ from one another because of the current branches.
3. PBADD field.
   The current branches of a given level are stored in a local high-density memory called *Level's Pattern Bank* (LPB), which serves all the machines in the same level. The pointer to this bank is in the PBADD bit field. If necessary, optimization of data storage can be achieved by addressing the LPB though a look-up table, the *Level's Address Bank* (LAB). PBADD consists of the address to the LPB of the previous level extended by few bits indicating which branch matched at that level. Of course, this information uniquely identifies the pattern block and the address to the LPB of the current level.

For each match, the matching machine holds the RB, PB, and PBADD fields for the next level, until the level controller pushes them in the down-FIFO. End event and start event words separate different events. These words are propagated down the pipeline through the FIFOs and used by the level controller to keep the correct event sequence: words of one event are pushed in the down-FIFO only after the words of the previous event.

B. Pattern Bank

In the TSP, there is no central memory containing all patterns. Rather, the pattern bank is delocalized on the different pipeline stages. Every LPB contains only the branches concerning its own level. The complete patterns of a given level can be resurrected only by properly combining the data stored in the LPBs of this and of all higher levels up to the root.

The LPB can be organized under the constraint that all data for a block test must be transferred to the matching machine in one clock cycle. If the TSP works for an n-layer road, a branch is an n-bit word. Therefore, a pattern block is identified by a subset of 2^n potential current branches. For instance, LPB could store 2^n-bit words, with each bit associated to one possible current branch: a value of "one" would indicate that the corresponding branch is actually consistent with a track candidate.

From the current branches of the LPB-word and from the parent branches of the up-FIFO PB-field, the matching machine decodes the patterns to be compared to the event stored earlier in a RB.

C. Tree Controller

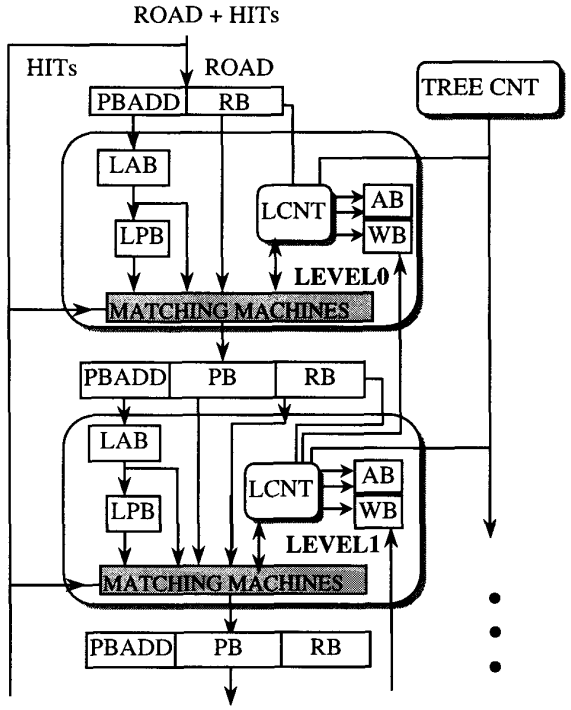

Figure 4: Tree controller and level controllers.

Figure 4 shows the level controllers and the tree controller. The tree controller is dedicated to broadcast the hits of incoming roads and to maintain a free buffer queue for the RBs. For each road received by the TSP an RB number is allocated to the road and a mechanism is necessary to free the buffer number upon completion of the road analysis.

To this purpose, every pipeline stage is equipped with two up-down counters for each RB: the Active-Block counter (AB) and the Waiting-Block counter (WB). For a given road buffer, AB monitors the number of blocks currently under test on the matching machines. As for WB, it monitors the number of blocks currently waiting in the down-FIFO. Each time a block is popped out of the up-FIFO, the level controller sends an up-pulse to the AB of its level and a down-pulse to the WB of the previous level. Each time the blocks generated by a matching machine are pushed in the down-FIFO, the level controller sends a down-pulse to the AB of its level and an up-pulse to the WB of the same level. The road analysis is complete when the counters of the corresponding road buffer reach zero on all levels. The tree controller watches all the counters and, upon

completion of a road analysis, it clears the RBs used at any level and it frees the buffer number in the buffer queue.

D. Match Logic

As explained, at any level of resolution the patterns consist of one bin per detector layer. The comparison of a pattern to the event is performed by independent logic circuits, which test in parallel the pattern bins on different layers.

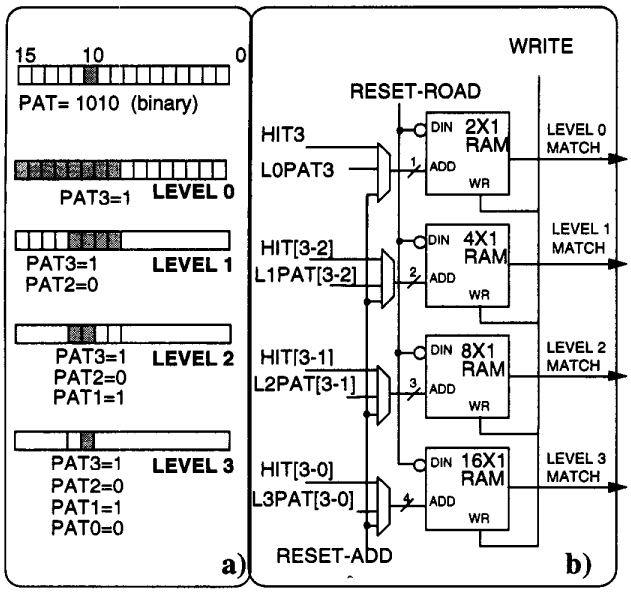

Figure 5: a) A pattern bin and its binary code at different levels of resolution, in a 16-bin layer. b) The match logic of a single layer implemented at different pipeline stages.

As an example, figure 5b shows how the matching circuits for one layer appear at each pipeline stage, for the case of 16-bin fat roads. At level n, counting from zero, the basic element is a 1-bit wide RAM with 2^{n+1} locations. These locations are in one-to-one correspondence with the layer bins at that level of resolution. By storing a logical "one" in every location corresponding to a hit bin, the RAM can be downloaded with a "snapshot" of the detector layer at the resolution appropriate to the level. To this purpose, when the TSP receives the road hits, the RAMs have all the data inputs set to "one" with the address lines driven by the HIT bus. Each layer hit received by the TSP is presented on the HIT bus while a write cycle is performed on the RAMs. For 16-bin fat roads, the HIT bus is 4-bit wide: HIT[3-0]. The logical OR of adjacent bins to meet the resolution at level-n is achieved by addressing the RAMs with the n+1 most significant HIT bits, e.g. HIT[3-2] at level-1. The RAMs implement the road buffers.

During the pattern comparison, the RAMs are used in read mode and addressed by the PATtern bus (PAT). Figure 5a shows how the most significant PAT bits identify one pattern bin with the resolution appropriate to the level. This is done in the same way the most significant HIT bits identify a hit bin during data writing. Each time the PAT bus addresses a RAM location corresponding to a hit bin, the match is flagged by reading out a logical "one" (layer match). The n+1 PAT bits used at level-n are generated in the following way. The n most significant bits come from the up-FIFO PB-field and identify the parent branches, while the remaining bit is decoded from the level's pattern bank and identifies the current branch.

The layer matches are monitored by a look-up table for global pattern match. The use of a look-up table allows implementation of any criteria, such as a majority logic that accounts for detector inefficiencies on some layers. In case of pattern match, the PAT bits used at a given level are put in the PB-field of the down-FIFO and passed to the next level.

When the road analysis ends, all RAMs are addressed by the RESET-ADD bus and cleared by cycling through every memory location.

Field programmable gate arrays, containing many small RAMs among their basic building blocks, are an ideal support for implementing the TSP match logic.

E. TSP Output

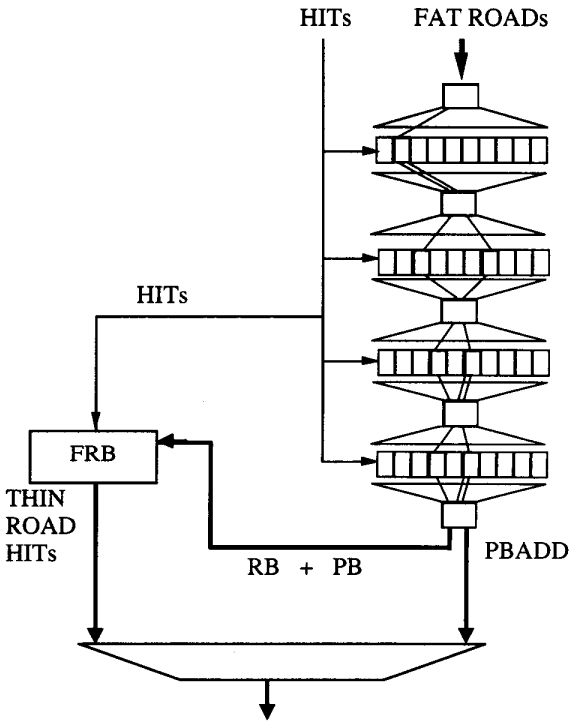

Figure 6: TSP output. The thin road is identified by the PBADD-field. The hits are retrieved from the FRB identified by the RB-field, at the address indicated by the PB-field.

Figure 6 shows a sketch of the TSP output. For each pattern match at the bottom level, TSP outputs the found thin road, which is identified by the PBADD-field in the last FIFO. In addition, TSP must also output the full resolution hits belonging to the thin road, for further processing. These tasks are performed by an output stage.

The output stage is provided with *Full resolution Road Buffers* (FRB). FRBs are numbered as the RBs of the standard pipeline stages and are subject to the same free buffer queue.

As the RBs, the FRBs are downloaded with full resolution hits when TSP receives the data. Unlike the RBs, which are 1-bit wide, the FRBs are RAMs wide enough to store all the hit bits, saving address space with respect to the RBs.

If TSP is designed to stop before reaching the full detector resolution, each thin road bin may contain multiple hits. Storing full resolution hits, and retrieving those belonging to a found road, is the function executed by the Hit Buffer [5] of the Silicon Vertex Tracker, in the Collider Detector at Fermilab. It is implemented in the same way by the TSP output stage. A great simplification comes from the fact that the FRBs must store only the hits of fat roads, while the Hit Buffer stores all hits of the entire detector. Essentially, the FRB addresses are grouped in sections; each section is dedicated to a single bin. When a thin road is found, its bins indicate which FRB sections must be read to fetch out the hits. The RB-field of the last pipeline FIFO gives the FRB ID, while the PB-field gives the road bins and, hence, the FRB sections.

IV. CONCLUSIONS

A powerful dedicated hardware is proposed for execution of a general algorithm based on a binary search strategy. Although developed for on-line trackers of high-energy physics, the device can be used in different applications where a large bank of patterns must be searched. The advantages in terms of speed come from the chosen search strategy, the strong parallelism, and the pipeline structure of the hardware.

V. REFERENCES

[1] M. Dell'Orso and L. Ristori, "A highly parallel algorithm for track finding", *Nucl. Intsr. and Meth.*, vol. A287, pp. 436-440, 1990.

[2] P. Battaiotto et al., "The Tree-Search Processor for Real Time Track Pattern Recognition", *Nucl. Intsr. and Meth.*, vol. A287, pp. 431-435, 1990.

[3] P. Battaiotto et al., "A Fast Track Finder for Triggering Applications in High Energy Physics", *Nucl. Intsr. and Meth.*, vol. A293, pp. 531-536, 1990.

[4] M. Dell'Orso and L. Ristori, "VLSI structures for track finding", *Nucl. Intsr. and Meth.*, vol. A278, pp. 436-440, 1989.

[5] S. Belforte et al., "The SVT Hit Buffer", *IEEE Trans. on Nucl. Sci.*, vol. 43, pp. 1810-1813, 1996.