FERMILAB-Conf-03/357-E November 2003

Hash Sorter - Firmware Implementation and an Application for the Fermilab BTeV Level 1 Trigger System

J. Wu, M. Wang, E. Gottschalk, G. Cancelo and V. Pavlicek

Abstract— **A hardware hash sorter for the Fermilab BTeV Level 1 trigger system will be presented. The hash sorter examines track-segment data before the data are sent to a system comprised of 2500 Level 1 processors, and rearranges the data into bins based on the slope of track segments. We have found that by using the rearranged data, processing time is significantly reduced allowing the total number of processors required for the Level 1 trigger system to be reduced. The hash sorter can be implemented in an FPGA that is already included as part of the design of the trigger system. Hash sorting has potential applications in a broad area in trigger and DAQ systems. It is a simple O(n) process and is suitable for FPGA implementation. Several implementation strategies will also be discussed in this document.**

Index Terms—**BTeV, Trigger, Hash Sorter, Firmware.**

I. Introduction

BTeV is a B-physics experiment that has been proposed [1] to run in the Tevatron at Fermilab.

The core of the BTeV detector is a 30 station Si-pixel inner tracker immersed in a 1.6 Tesla dipole field. There are over $20x10^6$ active rectangular pixels each measuring 50μ x 400μ. Each pixel station has two planes, one with narrow pixel dimension oriented in the x-direction, called "non-bend view", and the other with narrow dimension in the y-direction called "bend view". The coordinate data of particle hits measured by the pixel detector are sent to the Level 1 vertex trigger system.

The primary task of the Level 1 vertex trigger system [2-4] is to select events of interest for B-physics analyses by reconstructing charged tracks and primary interaction vertices and finding tracks detached from the primary vertices.

The track and vertex reconstruction is done in two phases. The first phase is executed in the segment processor. Hits on three adjacent silicon stations are linked together to form track segments, called "triplets". The segment processor forms two types of triplets, "internal" and "external", corresponding to the beginning and the end of a track in the pixel detector.

In the second phase, the triplet information is passed to the track and vertex processor, which consists of a farm of embedded processors. The farm matches "internal" with "external" triplets to form complete tracks. The track parameters are used to find primary vertices for tracks that appear to come from a common point in the beam region. The trigger decision is based on the presence of tracks that appear to be "detached" from a primary vertex, since this is a characteristic feature of B particles.

J. Wu, M. Wang and E. Gottschalk, G. Cancelo and V. Pavlicek are with Fermi National Accelerator Laboratory, Batavia, IL 60510 USA (phone: 630-840-8911; fax: 630-840-2950; e-mail: jywu168@ fnal.gov).

II. Principle and Implementation

A. Hash Sorting for $O(n^2)$ Algorithm Acceleration

Matching internal to external triplets is a time-consuming process. Each external triplet is checked against the entire list of internal triplets for possible matches. The process is an $O(n^2)$ algorithm. In the BTeV level 1 trigger baseline design, this process is done in C-code and takes significant portion of processing time.

A necessary condition for an internal triplet to match an external triplet is that their slopes in the non-bend view must be approximately equal. One possible solution is to sort the triplets into several bins based on their slopes. With sorted data, each external triplet will only need to be matched to internal triplets in one or two bins, rather than all. Therefore each external triplet will need only to be checked against a much smaller list resulting in a significant reduction of the processing time.

The sorting can be done in an FPGA before the data is sent to the embedded processor. We will discuss this aspect next.

B. Firmware Implementation

A test design has been implemented on the current pre-prototype version of the Level-1 Track and Vertex hardware. This hardware that forms the basic unit of the track and vertex farm consists of four TI 671x Digital Signal Processors (DSP's), three FPGA's and two microcontrollers. Triplet data from the segment processor is received by this hardware after going through an event-building switch. One of the FPGA's on this hardware, called Buffer Manager (BM), sends all the triplet data for one crossing to one of the four DSP's.

The hash sorting function is performed parasitically in the BM. One block named "DxBin" and four blocks named "HashBlk" (one for each DSP) are added into the BM FPGA.

The DxBin block simply calculates triplet parameters used for hash sorting. When the triplet data are filling the DATA RAM buffer for each DSP, the data lines "D" and address

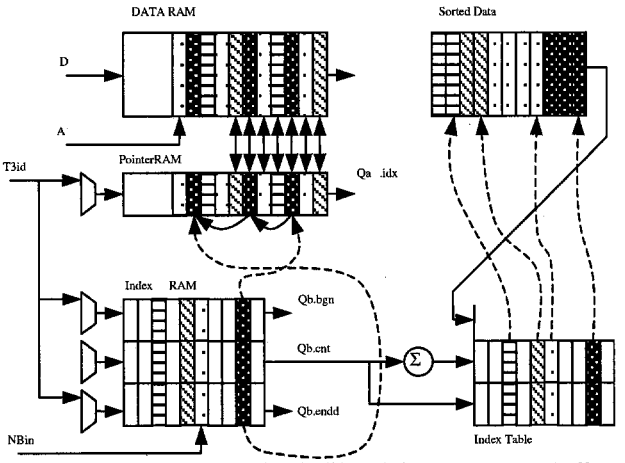

Fig. 1. The hash sorting block (HashBlk) and the DATA RAM buffer are shown. The index table and sorted data represent results after DMA.

lines "A" are monitored by the DxBin block. The hash bin number "NBin" is calculated based on the non-bend view slope of the triplet. The triplet ID, or serial number of the current triplet, "T3id", is also calculated, which is simply the higher bits of the DATA RAM address "A".

The hash sort block, HashBlk and the DATA RAM buffer are shown in Fig. 1. The core part of the HashBlk block consists of two memory areas, "PointerRAM" (Qa) and "IndexRAM" (Qb). The "PointerRAM" stores the pointers of triplets. The "IndexRAM" has three bit fields: "Qb.bgn", "Qb.cnt" and "Qb.endd", which represent the beginning location, the count and the end location of a hash bin. While the triplet data are filling the DATA RAM buffer, the hash bin number NBin and the triplet ID T3id associated with the triplet are used to fill the "PointerRAM" and the "IndexRAM" simultaneously. The following actions are performed in the process:

1. The contents of the IndexRAM at the location associated with the hash bin number NBin are read.
2. If the current triplet is not the first to be filled into the current hash bin, i.e., if the count of the bin "Qb.cnt" is not 0, the PointerRAM location, indexed by the "Qb.endd" will store the triplet ID, T3id.
3. If the triplet is the first one in the current bin, the ID of the triplet T3id is written into both the "Qb.bgn" and "Qb.endd" field. Otherwise, "Qb.bgn" will be kept unchanged and only "Qb.endd" will be update with T3id. In either case, "Qb.cnt" will be incremented by 1.

The actions above use only 3 clock cycles, given the available time of 4 clock cycles for filling the 4 32-bit data words of each triplet. After these actions, single directional link lists of hash bins are formed in the PointerRAM available for the later stages to use--data are logically hash sorted.

In the next step, data will be downloaded into the DSP via a direct memory access (DMA) process. The data are further physically sorted during the download.

An index table is first dumped into the DSP. The index table will provide the beginning location and the count of the triplets in the sorted data block for each hash bin so that the Level 1 trigger software can access the sorted data efficiently. The table is very ease to produce in fly during DMA.

After downloading the index table, the triplet data are sent out via DMA. The order of presenting the triplet data to the output port is controlled by the link lists stored in the PointerRAM. After DMA, the data stored in the DSP memory become physically hash sorted. The entire download process, including downloading both index table and triplet data block, is completed in a single DMA process. After the DSP started the DMA, no further software intervention is needed.

It should also be pointed out that the PointerRAM and IndexRAM are actually implemented in different area of one physical dual-port RAM block. The process functions of the hash sorting have been so adjusted that this kind of implementation option can be realized. In modern FPGA devices, dual-port RAM blocks are commonly available while the number of blocks per device is still limited. So it is a useful design practice to combine the memories into a single memory block.

III. Silicon Resource Usage

The hash sorter described above was compiled in our current "Buffer Manager" FPGA device (Xilinx [5] xc2v1000). The following table shows the silicon resource usage reported by the compiler:

Resource	Total available	Used in DxBin	Used in HashBlk
Slices	5120	52 (1%)	72 (1%)
Slice Flip Flops	10240	73 (1%)	51 (1%)
4-input LUT's	10240	35 (1%)	75 (1%)
Block RAM's	40	0	1 (2%)

The entire hash sorter uses 1 DxBin block and 4 HashBlk blocks. Then the total logic cell usage is about 7% and the memory block usage is about 10% in the device.

IV. Timing Result

Computation times (in CUP clock cycles) of the Level 1 vertex trigger algorithm for the BTeV baseline design are measured to study the acceleration effects with hash sorting. Simulated events with 2 to 11 interactions per beam crossing are fed to the trigger algorithm. Each of the measurement points shown in Fig. 2 were obtained by averaging the total execution time of the trigger algorithm on roughly 2500 simulated beam crossings.

Two sets of measurements are made. The first set uses unsorted triplet data and the second uses hash sorted triplet data. One can see that the segment matching process has been improved by a factor of 4 to 4.5 by using hash sorted triplet data. For comparison, we also plotted the total trigger process times (that includes segment matching, track processing and vertex finding). One can see that the segment matching process takes a large portion of the entire process.

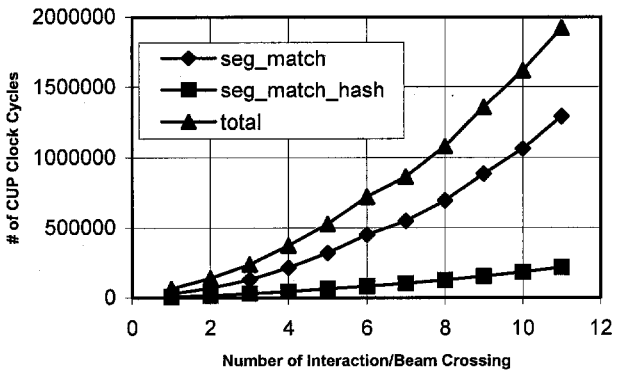

Fig. 2. Numbers of clock cycles needed to process triplet data in a beam crossing are plotted. The triangle points are total clock cycles to process unsorted data. The diamond and square points are ones of segment matching with unsorted or hash sorted data.

The timing measurement was performed using a 1.13 GHz Pentium III-M processor. Although the fast processors that we will choose in the future will supersede the absolute CPU time, the relative CPU clock cycles should exhibit a similar trend of acceleration with hash sorting.

V. Discussion

In computing algorithm, "hashing" is a process to convert a key K of a data item into a number:

$$m = h(K), \quad 0 \le m < M$$

where h(K) is called hash function and M is the total number of storage locations. The data item is then stored in the location indexed by m for future use. The hash function can be in any form, although remainder modulo a prime number M is chosen in many textbooks [6-7].

The process described early in this paper can be viewed as a special case of hashing process. The hash function we choose is binning just as the one when we book histogram. When the variable or key K and bin size are properly chosen, the function is simply shifting and masking out of lower bits.

The process can also be viewed as a special case of "radix sorting/searching" or "bucket sorting/searching". In some aspect, it can be visualized as an extension of histogram booking too. We choose the term "hash sorting" to emphasize the fact that each storing or recovering of a data item is a single memory access.

Regular sorting is an algorithm of at least $O(n \log n)$, while hash sorting is $O(n)$, with unsorted items within a bin. In many applications, the absolute order between two close up items cannot be well defined due to measurement errors on the variable or key K that sorting algorithm is based on. In this case, the result of hash sorting is not any worse than the regular sorting. However, the simplicity of hash sorting clearly becomes favorable comparing with the complexity of regular sorting.

Hash sorting can be implemented either in software or hardware. In hardware implementation, the only necessary function is to fill the "IndexRAM" and the "PointerRAM". The data items are already logically hash sorted once the single directional link lists are established in these memory locations. The physical order of the data items is further rearranged during the DMA in our example. In other applications, it may not be absolutely necessary. The rearrangement is just to provide some convenience for the software in the later stages. In fact, the rearrangement does not cost extra process time.

References

[1] BTeV Collaboration (May 2000). BTeV Proposal [Online]. Available: http://www-btev.fnal.gov/public/hep/general/proposal

[2] E. E. Gottschalk, "Detached Vertex Trigger," *Nucl. Inst. Meth*, vol. A 473, p. 167, 2001.

[3] E. E. Gottschalk, "BTeV DAQ and Trigger System – Some Throughput Usability and Fault Tolerance Aspects," in *Proc. CHEP 2001*, Beijing, 2001, p. 628.

[4] M. H. L. S. Wang, "BTeV Level 1 Vertex Trigger," *Nucl. Inst. Meth*, vol. A 501, p. 214, 2003.

[5] *Xilinx FPGA Data Sheets* [Web Site]. Available: http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp

[6] D. E. Knuth, *The Art of Computer Programming*, 2nd ed., vol. 3. Addison-Wesley, 1997.

[7] R. Sedgewick, *Algorithms*, 2nd ed., Addison-Wesley, 1988.