First evaluation of the CPU, GPGPU and MIC architectures for real time particle tracking based on Hough transform at the LHC

2014 JINST 9 P04005

(http://iopscience.iop.org/1748-0221/9/04/P04005)

PUBLISHED BY IOP PUBLISHING FOR SISSA MEDIALAB

RECEIVED: *November 4, 2013*
ACCEPTED: *February 15, 2014*
PUBLISHED: *April 7, 2014*

First evaluation of the CPU, GPGPU and MIC architectures for real time particle tracking based on Hough transform at the LHC

V. Halyo.[a,1] **P. LeGresley,**[a] **P. Lujan,**[a] **V. Karpusenko**[b] **and A. Vladimirov**[b]

[a] *Princeton University, Princeton, NJ, U.S.A.*
[b] *Colfax International, Sunnyvale, CA, U.S.A.*

E-mail: vhalyo@gmail.com

ABSTRACT: Recent innovations focused around *parallel* processing, either through systems containing multiple processors or processors containing multiple cores, hold great promise for enhancing the performance of the trigger at the LHC and extending its physics program. The flexibility of the CMS/ATLAS trigger system allows for easy integration of computational accelerators, such as NVIDIA's Tesla Graphics Processing Unit (GPU) or Intel's Xeon Phi, in the High Level Trigger. These accelerators have the potential to provide faster or more energy efficient event selection, thus opening up possibilities for new complex triggers that were not previously feasible. At the same time, it is crucial to explore the performance limits achievable on the latest generation multi-core CPUs with the use of the best software optimization methods. In this article, a new tracking algorithm based on the Hough transform will be evaluated for the first time on multi-core Intel i7-3770 and Intel Xeon E5-2697v2 CPUs, an NVIDIA Tesla K20c GPU, and an Intel Xeon Phi 7120 coprocessor. Preliminary time performance will be presented.

KEYWORDS: Particle tracking detectors; Trigger concepts and systems (hardware and software); Trigger algorithms

[1] Corresponding author.

doi:10.1088/1748-0221/9/04/P04005

Contents

1 Introduction **1**

2 Physics motivation **2**

3 Processor architecture **3**
- 3.1 Multi-core CPUs 3
- 3.2 Graphics Processing Units 3
- 3.3 MIC architecture (Intel Xeon Phi coprocessors) 4

4 Hough transform algorithm **4**

5 System description **6**
- 5.1 CPU architecture 6
- 5.2 NVIDIA GPU architecture 6
- 5.3 Intel coprocessor architecture 6

6 Optimization **7**
- 6.1 Optimization for Tesla GPU 7
- 6.2 Optimization for Intel i7 and Xeon CPUs and for Intel Xeon Phi coprocessors 7

7 Preliminary performance results **8**

8 Discussion **8**

9 Summary **10**

2014 JINST 9 P04005

1 Introduction

Scientific computing is a critical component of the LHC experiment, affecting detector operations, trigger, simulation, analysis, and the LHC computing grid. The trigger systems are especially sensitive to computing performance, as they must balance the physics goals of the experiment with the need to be able to process events in real time. Improvements in the speed with which the trigger system can reconstruct events can thus extend the physics reach of the LHC. One way this can be achieved is to take advantage of the flexibility of the trigger system by integrating coprocessors based on Graphics Processing Units (GPUs) or the Many Integrated Core (MIC) architecture into its server farm. Parallel processing will provide means not only to accelerate existing algorithms, but also the opportunity to develop new algorithms that select events that could have previously evaded detection.

These parallel algorithms will be able to reconstruct in real time all the trajectories of charged particles in the silicon tracker. It is well-known that the problem of track reconstruction becomes exponentially more difficult using traditional reconstruction algorithms as the number of hits in the detector increases due to high pileup. The new tracking can not only overcome the challenges faced by the trigger at high pileup, but it allows us to develop complex triggers that could select in real time final state topologies not possible before.

Leveraging processors derived from consumer products helps minimize costs associated with purchasing and operating large computing installations, such as those required for the LHC, but there is also a software aspect that needs to be considered. Parallel processors, in the form of multi-core CPUs and, more recently, highly programmable Graphics Processing Units (GPUs), may require a significant rethinking of algorithms and their implementations. At the same time, the High Energy Physics (HEP) community is at a crossroads with tentative confirmation of the Higgs particle completing the search for particles predicted by the Standard Model. The search for beyond the Standard Model (BSM) physics will require development of advanced algorithms for detecting rare new physics phenomena, and these new algorithms will need to be designed and implemented based on knowledge of the current state and likely future directions for processor architecture.

In this article, a new tracking algorithm based on the Hough transform will be evaluated for the first time on an Intel Xeon Phi and multi-core Xeon CPUs, and compared with performance on an NVIDIA Tesla K20c GPU. Preliminary time performance will be presented.

2 Physics motivation

Various BSM extensions predict the existence of new, strongly interacting particles that lead to final states with high jet multiplicities [1–6]. Other exotic models might include boosted jets [7, 8], R-parity-violating SUSY models with long-lived neutralinos [9, 10], long-lived neutral particles decaying at macroscopic distances from the primary vertex [11–13] in the tracker, quirks (also known as iquarks) [14], displaced black holes [15], or long-lived Higgs particles [11, 13]. The key to observing these events [16] is a fast tracking algorithm executed in real time in the trigger system that allows us to reconstruct all prompt (i.e., tracks that originate from the interaction point) and non-prompt tracks for further careful, offline analysis.

Both ATLAS [17] and CMS [18] use a standard multi-level trigger system, with the lowest level (L1) using fast and simple criteria implemented in hardware and firmware, and higher-level triggers which reconstruct the full event in software and can apply selection criteria similar to those used offline, using a farm of commercial CPU processors. This selection is designed to select only the events which contain interesting physics processes. However, the need to process the events in real time imposes very stringent limits on the complexity of the analysis that can be done at the trigger level. For instance, in the offline reconstruction at CMS, tracks can be reconstructed even if they originate relatively far away from the primary interaction point. However, reconstructing these tracks is computationally expensive, as the number of possible combinations is quite large; hence, displaced tracks are not reconstructed in the CMS high-level trigger (HLT) and so one cannot directly trigger on events which contain such a signature.

The flexibility of the trigger system makes it amenable to adding new kinds of hardware, such as GPU- or MIC-based coprocessors. Thus it is easy to integrate the hardware necessary

2014 JINST 9 P04005

for the development of massively parallel algorithms that will not only reconstruct in real time all the particle trajectories in the event, but allow the development of new algorithms that can select final state topologies previously not possible in the trigger. This article will evaluate the most computationally expensive application of the tracking algorithm for the first time on various hardware architectures. These benchmarks are critical for any future LHC trigger upgrade.

3 Processor architecture

In the past decade, it has become apparent that it is no longer possible to rely solely on increases in processor clock speed as a means for extracting additional computational power from existing architectures. The underlying reasons are complex, but center around reaching what may be fundamental limitations in semiconductor device physics. For this reason, recent innovations have focused around *parallel* processing, either through systems containing multiple processors, processors containing multiple cores, and so on.

3.1 Multi-core CPUs

Traditional CPUs have been evolving in recent years mainly through the growth of parallelism rather than increasing clock frequencies. Multiple cores on a single chip have been introduced; the Non-Uniform Memory Access (NUMA) architecture was incorporated, which allows transparent use of multiple CPU sockets within a single shared memory system; and vector processing units have evolved to new instruction sets and longer vector registers. At the same time, the architecture of CPUs has become more efficient with deeper pipelines, smarter branch prediction and hardware prefetching of data from memory into caches.

Generally speaking, the CPU architecture is resource-rich and is capable of running workloads with complex and non-local memory access patterns, with or without task and data parallelism. However, when data locality, task parallelism (multi-threading) and data parallelism (vectorization) are present in the application, the potential performance benefits become tremendous. For instance, the AVX extensions (Advanced Vector Extensions) available in modern CPUs offer the ability to quickly operate on floating-point vectors. Optimization for such CPU systems poses considerable challenges; however, it also promises significant benefits.

3.2 Graphics Processing Units

One interesting technology which has continued to see exponential growth is graphics processing. A modern Graphics Processing Unit (GPU) is a massively parallel processor with thousands of execution units to handle highly parallel workloads related to computer graphics. By making these execution units highly programmable, manufacturers have made the massive computational power of a modern GPU available for more general-purpose computing, as opposed to being hard-wired for specific graphical operations. In certain applications that can be executed in massively parallel fashion, this can yield several orders of magnitude better performance than a conventional CPU. Manufacturers have taken advantage of these possibilities to release GPUs designed for general-purpose computing, such as the NVIDIA Tesla line. To facilitate the use of general-purpose GPU computing, NVIDIA has also developed CUDA ("Compute Unified Device Architecture") [19], which allows rapid development of GPU software in nearly standard C code.

2014 JINST 9 P04005

3.3 MIC architecture (Intel Xeon Phi coprocessors)

With the growing success of GPUs as a massively parallel processor well suited to more general purpose applications, Intel has introduced a line of products based on a Many Integrated Core (MIC) architecture, marketed under the name Xeon Phi. The Xeon Phi coprocessors are symmetric multiprocessors; they physically look similar to a GPU in that they plug into a host system via PCI Express. They run a μOS Linux Operating System. However, a Xeon Phi coprocessor cannot be used as a stand-alone processor, and requires a host system to operate.

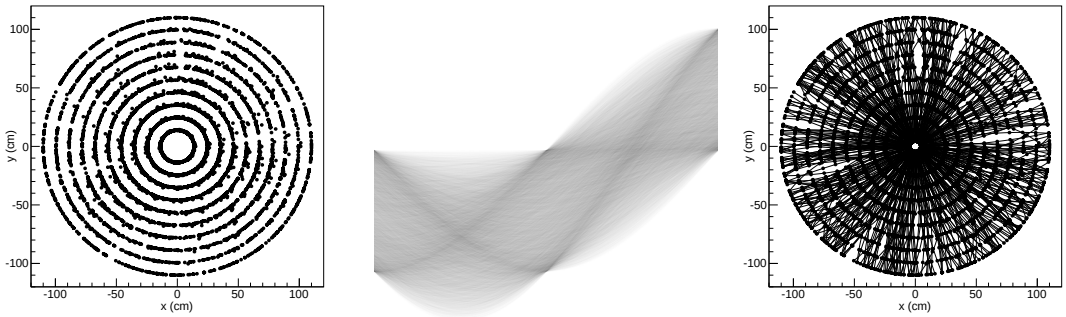
From an architectural perspective, they also have many similarities to GPUs. Xeon Phi is based on an x86 Pentium core architecture from the early 1990s. This is a much simplified, and hence smaller, core compared to modern x86 CPUs. To make these simplified cores more computationally powerful, 512-bit-wide vector units have been added to the core. Because of the simplified nature of the core each coprocessor features 60+ cores clocked at 1 GHz or more, supporting 64-bit x86 instructions. The exact number of cores depends on the model and the generation of the product. These in-order cores support four-way hyper-threading, resulting in more than 240 logical cores. The cores of an Xeon Phi coprocessor are interconnected by a high-speed bidirectional ring, which unites the L2 caches of the cores into a large coherent aggregate cache over 30 MB in size. The coprocessor is equipped with 6 to 16 GB of on-board GDDR5 memory. The speed and energy efficiency of Xeon Phi coprocessors come from their vector units. Each core contains a vector processing unit (VPU) with 512-bit SIMD (Single Instruction Multiple Data) vectors supporting a new instruction set called Intel Initial Many-Core Instructions (Intel IMCI). The Intel IMCI include, among other instructions the Fused Multiply-Add (FMA), reciprocal, square root, power and exponential function operations, commonly used in physical modeling and statistical analysis. The theoretical peak performance of an Xeon Phi coprocessor is 1 TFLOP/s in double precision. This performance is achieved at the same power consumption as in two Xeon CPU processors, which yield up to 2-3 times less GFLOP/s.

The greatest difference between the MIC architecture and GPUs is that it is possible to develop and optimize a single code in C, C++ or Fortran to use on both a multi-core CPU and on a Xeon Phi coprocessor. In contrast, GPU applications based on the CUDA architecture are built from a code that is different from the CPU application code, in terms of both syntax and algorithm. This programming model continuity of the MIC architecture is attractive when the application is developed not just for the accelerator, but for a heterogeneous system composed of CPUs or coprocessors, or when the application is deployed to clients which may or may not have access to a coprocessor.

4 Hough transform algorithm

As a demonstration of how computationally intensive 2D tracking algorithms can take advantage of massively parallel GPU processing, the authors developed an implementation of the Hough transform using CUDA [16] in order to measure in real time the transverse momentum of prompt or non-prompt tracks. The Hough transform [20–22] is an image processing algorithm for feature detection that considers all possible instances of a parameterized feature such as a line or circle. Each possible instance of a feature starts with zero votes in the parameter space, and then for each piece of input data votes are added to the feature instances that would include that input data. After

2014 JINST 9 P04005



(a) (b) (c)

Figure 1. Hough transform algorithm applied to a simple example. Left: hits in a simulated event with curved tracks. Centre: each hit results in a curve of votes in parameter space. Locations with many votes are likely to be tracks in the original data. Right: candidate tracks identified from finding local maxima in the parameter space.

all input data has been processed the votes in the parameter space are processed. Locations in the parameter space with more votes are likely to be actual features in the input data so this step amounts to looking for local maxima in the parameter space. Once candidate features have been identified, more expensive computations can be applied to confirm the existence of the feature. One should note that the Hough transform approach as presented above is computationally expensive. Hence, simplification of the standard Hough transform method [23, 24] had to be used in the past for the purpose of track finding in high energy physics [25, 26].

In practice, the Hough transform is implemented using a discrete grid in the parameter space. In our case, the parameter space consists of two variables — the track curvature ρ and the azimuthal angle ϕ — and the parameter space is discretized into a 2048x2048 grid, which offers a balance between calculation speed and precision. The first step then consists of finding and incrementing the appropriate grid spaces (voxels) in parameter space for a given hit, and the second step consists of searching for local maxima on the grid.

Figure 1 shows the algorithm in operation for a simple case with 500 curved tracks. The simulated curved track data shown in figure 1(a), can correspond to many different possible tracks in parameter space as shown in figure 1(b). When integrated over the entire data set, peaks appear in parameter space at the values corresponding to the actual, physical trajectories as shown in figure 1(c).

One clear advantage of the Hough transform over the combinatorial track finding algorithms that are currently standard is that the execution time is linear with respect to the number of hits present in the event, while combinatorial algorithms, as the number of combinations increases much more rapidly with respect to the number of hits, show a worse dependence on the number of hits. In addition, the Hough transform is naturally tolerant of missing hits or hits that do not exactly fit the candidate features, due to limited resolution or the discretization used in computing the parameter space. While the Hough transform is more expensive to implement for a small number of hits, the ability to take advantage of multithreading offers the chance of significantly improved performance for events with a large number of tracks.

2014 JINST 9 P04005

One should note that this paper does not imply that executing machine vision and pattern recognition algorithms is necessarily more appropriate for parallel processing compared to the conventional combinatorial algorithms used in high energy physics. Rather, it is a complementary way that the authors began studying due to the similarity of the problem with computer image analysis. These techniques have proven successful in image processing using the Hough transform on GPUs.

In the following studies, the sample input data was generated using a Monte Carlo simulation of a simple detector model where only the transverse plane is considered. The model contains a simulated beam pipe with a radius of 3.0 cm surrounded by ten concentric, evenly-spaced tracking layers with an overall radius of 110.0 cm. A hit resolution of 0.4 mm in each direction is used, corresponding to a relative p_T resolution of approximately 7% at 100 GeV/c.

In previous studies, we have presented preliminary results using a GPU implementation of the Hough transform [15, 16]. This paper shows the first results of an implementation on the Xeon CPU and Xeon Phi architecture, along with improvements in the GPU implementation.

5 System description

5.1 CPU architecture

The CPU performance for the Intel Xeon processor is measured on a Colfax CX2265i-XP5 server [27] with 128 GB of 1600 MHz DDR3 ECC memory, based on a two-socket Intel Xeon E5-2697v2 (Ivy Bridge) CPU for servers. The thermal design power (TDP) of each CPU socket is 130 W. The sockets are interconnected by a Quick Path Interconnect (QPI) link and forming a shared-memory NUMA system. Each socket has 12 physical cores (24 cores in the system) clocked at 2.7 GHz (turbo frequency of 3.5 GHz) with two-way hyper-threading. The vector units of the system support the AVX instruction set with 256-bit vector registers.

The host operating system is CentOS 6.4 Linux with kernel version 2.6.32-358.11.1.el6.x86_64. The code was compiled with the Intel C++ compiler version 13.1.3.

The CPU performance for the quad-core Intel i7-3770 processor is measured on a different desktop system, which is also the host of the NVIDIA Tesla GPU. The TDP of the i7-3770 CPU is 77 W.

5.2 NVIDIA GPU architecture

The NVIDIA Tesla GPU model used for the results presented in this paper is the Tesla K20c (active-cooled model for workstations), which contains 2496 cores clocked at 706 MHz, 5 GB of on-board memory with a clock of 2.6 GHz, and a 320-bit memory interface. The code was written in CUDA C and compiled with the nvcc compiler, version 5. The host operating system is CentOS 6.4 Linux with kernel version 2.6.32-358.11.1.el6.x86_64.

5.3 Intel coprocessor architecture

The Xeon Phi coprocessor performance is measured on the same system as the CPU performance. The system contains one Xeon Phi coprocessor of the QS-7120P series (passive-cooled model for servers) with 61 cores at 1.33 MHz, C0 coprocessor stepping, and 16 GB of GDDR5 RAM at 2750 MHz. The driver stack is MPSS version 2.1.6720-13.

2014 JINST 9 P04005

6 Optimization

A time profile of the tracking algorithm shows that 92% of the algorithm execution time is spent on executing the Hough transform itself. Therefore, we optimize the Hough transform separately on three different architectures.

6.1 Optimization for Tesla GPU

Extensive work had already been done to implement and optimize the Hough transform for the NVIDIA GPU architecture [15, 16]. These past optimizations included minimizing expensive trigonometric functions, developing an efficient memory access pattern for reading and writing global memory, and safely handling updates of values in the parameter space to avoid race conditions. Additional performance optimizations have been undertaken to further improve performance, including:

- additional reductions in global memory accesses;
- an additional optimization of the global memory access pattern to maximize efficiency of memory coalescing;
- replacement of any remaining atomic memory accesses to global memory with atomic memory accesses to shared memory.

The global memory optimizations are particularly important because it is implemented as off chip Dynamic Random Access Memory (DRAM) and is much more costly to access (in terms of both latency and bandwidth) compared to on chip memory locations. However, the use of global memory is unavoidable because it is the only memory location available for storage of data that needs to persist over a sequence of computational operations on the GPU.

6.2 Optimization for Intel i7 and Xeon CPUs and for Intel Xeon Phi coprocessors

Although Hough transform implementations exist in standard libraries such as the Intel Performance Primitives (IPP) library, these do not take advantage of multi-threading. For that reason, in this work, we created our own parallel implementations and optimized them to extract the best performance out of the parallel compute devices that we benchmarked.

Because of the similarities of Intel i7 and Intel Xeon CPUs with Xeon Phi coprocessors, we developed a single code for all three platforms. The executables for CPUs and coprocessors are not binary-compatible, and the C++ code must be compiled twice: once for the CPU and another time for the coprocessor. We used preprocessor macros to insert different values of tuning parameters into the CPU and the MIC architecture binaries.

Other than the tuning parameters mentioned above, the optimization methods for both platforms are common:

- thread parallelism was implemented in the OpenMP framework [28] for the outer loops of the Hough transform and local maxima search;
- where possible, the code was written in such a way that the compiler is able to perform automatic vectorization;

2014 JINST 9 P04005

- cross-thread synchronization was avoided by the use of the OpenMP reduction facility and thread-private data storage;
- where synchronization was necessary (e.g., for merging thread-private storage into a shared storage array), improved synchronization methods were used, such as ordered for-loops in OpenMP;
- strip-mining and blocking techniques (see, e.g., [29] and references therein) were applied to nested loops in order to improve the data access locality. These techniques change the order of operations in nested loops, so that data loaded into caches is re-used as soon as possible, which reduces the frequency of long-latency memory accesses;
- in order to utilize the coprocessor, data was moved from the CPU to the coprocessor using the offload functionality provided by the Intel C++ compiler;
- data persistence on the coprocessor was used to avoid a reallocation penalty when multiple frames are offloaded and analyzed.

Thanks to the NUMA architecture, a two-socket system acts as a single CPU from the programming perspective, with OpenMP threads seamlessly scaling across both sockets. Therefore, no additional development was required to utilize two CPU sockets. However, in order to bind software threads to the respective core caches, we enforced thread affinity, which prevented software thread migration across logical cores. This was done by setting the environment variable `KMP_AFFINITY=compact,granularity=fine` prior to running the benchmarks.

7 Preliminary performance results

Figure 2 shows the resulting time performance as a function of the number of tracks in the event for four computing platforms: an Intel i7-3770 CPU, a dual-socket Intel Xeon E5-2697v2 CPU system, an NVIDIA Tesla K20c GPU, and an Intel Xeon Phi 7120P coprocessor. The reason for comparing the accelerators to a dual-socket CPU system (rather than single-socket) is that the TDP of the GPU and of the coprocessor is 225 and 300 W, respectively, whereas a single CPU chip is rated at 130 W. Therefore, two CPUs make for a more meaningful watt-to-watt comparison.

According to figure 2, the GPU performs almost as fast as dual Xeon CPUs for all event sizes. The Xeon Phi coprocessor and the i7 CPU are considerably slower. For large events (5000 tracks), the Xeon Phi is 3x slower, and the i7 is 5x slower than the Xeon system. The 5x ratio of the Xeon to i7 performance is equal to the ratio of the numbers of cores in these platforms corrected for the difference in their clock frequency. For small events (fewer than 200 tracks), the processing time reaches a plateau at a level that is is almost an order of magnitude greater for Xeon Phi and i7 than for Xeon and the GPU.

8 Discussion

The Hough transform calculation is a highly parallel task. However, it cannot take advantage of the GPU or many-core architecture in an optimal way because of the nature of this calculation.

2014 JINST 9 P04005

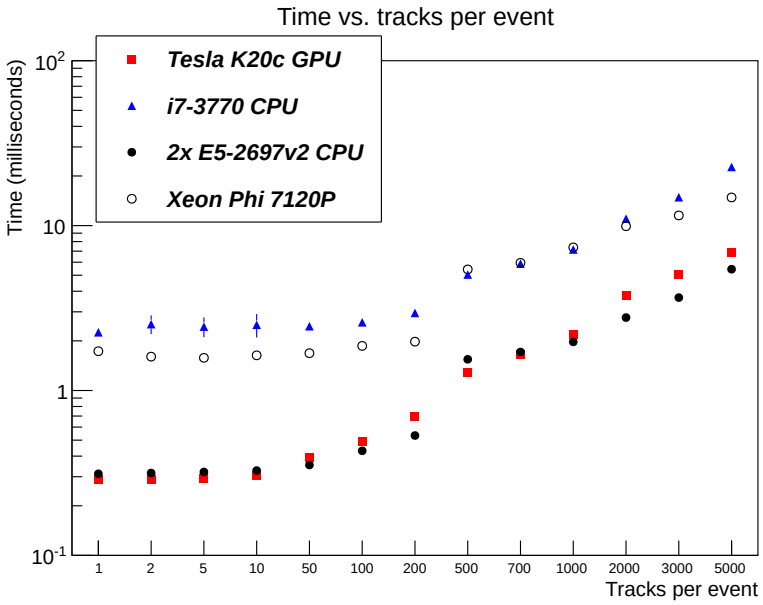


Figure 2. Performance of the Hough transform algorithm on four platforms: NVIDIA Tesla K20c GPU (red squares), Intel i7-3770 CPU (blue triangles), dual-socket Intel Xeon E5-2697v2 CPU (solid black circles), and Intel Xeon Phi 7120P coprocessor (open black circles), as a function of the number of simulated tracks in the event. A 2048x2048 grid is used in the parameter space. The error bars indicate variations between successive runs, which are negligibly small in most cases.

The first part of the calculation counts the number of points in the voxels of (ρ, θ) space. This operation has indirect memory accesses in the form `A[B[i]]+=1`, i.e., the data are written to an array index, which itself is looked up from another array. Due to a stochastic pattern of memory access, this operation cannot be performed with streaming memory accesses, which is a sub-optimal operation regime for the GPU and the MIC architecture. On the GPU this is mitigated by replacing accesses to global memory with accesses to on-chip shared memory. For the Xeon Phi coprocessor, in the absence of hardware prefetching for the Level 1 cache, performance in cases like this is dependent on software prefetching. Future versions of the Intel C++ compiler may be able to automatically implement software prefetching for indirect array accesses and improve the performance [30].

The second part of the calculation, where the transformed data set is searched for lines, has a more regular memory access pattern; however, a considerable part of this calculation uses scalar instructions and a non-trivial reduction pattern. This is also a sub-optimal workload for the GPU and the MIC architecture.

GPU and Xeon Phi architecture are designed as computing accelerators and have a higher theoretical peak memory bandwidth and arithmetic performance than the Xeon or i7 CPU. However, for this application, they do not provide significant acceleration compared to the multi-core Xeon processor. In fact, the Xeon Phi coprocessor is even falling behind in performance behind the Xeon processor of comparable TDP by a significant factor. This result is expected, because as mentioned above, the nature of the problem does not allow the GPU and the Xeon Phi coprocessor to efficiently exploit their architecture. At the same time, the Xeon CPU has a resource-rich architecture,

2014 JINST 9 P04005

which compensates for sub-optimal memory and operations traffic with its large unified Level 2 cache, hardware prefetchers for the Level 1 and Level 2 caches, higher clock frequencies of the cores, and shorter vector units.

Nevertheless, even at the currently observed performance, the inclusion of a GPU or a Xeon Phi coprocessor into the system configuration may be justified in some cases:

a) If the calculation is offloaded to the accelerator (GPU or coprocessor), the host CPU is free to do other tasks, such as preparation or post-processing of data.

b) When it is desirable to achieve the best possible performance within a single chassis, or within a limited rack space in a cluster, it is possible to use compute nodes with up to eight accelerators (GPUs or Xeon Phi coprocessors) such as [31, 32]. In addition, a Xeon CPU may be used simultaneously with accelerators to process a part of the workload. Programming models for the CUDA and the Xeon Phi architecture allow easy implementation of heterogeneous work sharing between CPUs and multiple accelerators (see, e.g., [33] and [34]).

c) Finally, in cases where a cluster of compute nodes is deployed for solving the problem of track detection in parallel, it may be advantageous to ramp up performance by equipping the compute nodes with accelerators instead of increasing the number of compute nodes. The accelerator approach may lower the initial set-up costs and the operational costs, because the number of server boards, network switches, the rack space requirements and power consumption may be lower than with a CPU-only approach. Power consumption estimates are beyond the scope of this work. The same applies to equipment costs; however, prices are available in the public domain and via commercial quotes.

One should also note that the relative momentum resolution of the tracks obtained using the Hough transform is not yet as good as that obtained using current HEP tracking [35]. In order to match the curent resolution, one would need to approximately double the time performance results obtained in figure 2.

9 Summary

Parallel processing provides great promise for enhancing the performance of the trigger at the LHC and extending its physics program. However, to achieve this goal, significant rethinking of the hardware and algorithms have to be considered. These decisions impact initial acquisition and setup costs, rack space, power limitations, operational costs associated with running the hardware and maintaining the software, etc. In this article, the authors optimized a real time tracking algorithm based on Hough transform algorithm on four different architectures Tesla GPU, Xeon and i7 CPUs, and Xeon Phi coprocessor. The outcome provides a benchmark that may be used to identify the optimal computing system configuration for the required throughput, cost and other compliance factors of the Hough transform based tracking detector. These results suggest that using an optimized algorithm on hybrid systems with add on coprocessors could be the leap necessary to discover new physics at the LHC.

2014 JINST 9 P04005

References

[1] R.S. Chivukula, M. Golden and E.H. Simmons, *Six jet signals of highly colored fermions*, *Phys. Lett.* **B 257** (1991) 403.

[2] R.S. Chivukula, M. Golden and E.H. Simmons, *Multi-jet physics at hadron colliders*, *Nucl. Phys.* **B 363** (1991) 83.

[3] E. Farhi and L. Susskind, *A Technicolored G.U.T.*, *Phys. Rev.* **D 20** (1979) 3404.

[4] W.J. Marciano, *Exotic New Quarks and Dynamical Symmetry Breaking*, *Phys. Rev.* **D 21** (1980) 2425.

[5] P.H. Frampton and S.L. Glashow, *Unifiable Chiral Color With Natural Gim Mechanism*, *Phys. Rev. Lett.* **58** (1987) 2168.

[6] P.H. Frampton and S.L. Glashow, *Chiral Color: An Alternative to the Standard Model*, *Phys. Lett.* **B 190** (1987) 157.

[7] J. Thaler and L.-T. Wang, *Strategies to Identify Boosted Tops*, *JHEP* **07** (2008) 092 [arXiv:0806.0023].

[8] A. Altheimer et al., *Jet Substructure at the Tevatron and LHC: New results, new tools, new benchmarks*, *J. Phys.* **G 39** (2012) 063001 [arXiv:1201.0008].

[9] L.M. Carpenter, D.E. Kaplan and E.-J. Rhee, *Reduced fine-tuning in supersymmetry with R-parity violation*, *Phys. Rev. Lett.* **99** (2007) 211801 [hep-ph/0607204].

[10] P.W. Graham, D.E. Kaplan, S. Rajendran and P. Saraswat, *Displaced Supersymmetry*, *JHEP* **07** (2012) 149 [arXiv:1204.6038].

[11] M.J. Strassler and K.M. Zurek, *Discovering the Higgs through highly-displaced vertices*, *Phys. Lett.* **B 661** (2008) 263 [hep-ph/0605193].

[12] M.J. Strassler and K.M. Zurek, *Echoes of a hidden valley at hadron colliders*, *Phys. Lett.* **B 651** (2007) 374 [hep-ph/0604261].

[13] V. Halyo, H.K. Lou, P. Lujan and W. Zhu, *Data driven search in the displaced $b\bar{b}$ pair channel for a Higgs boson decaying to long-lived neutral particles*, *JHEP* **01** (2014) 140 [arXiv:1308.6213].

[14] K. Cheung, W.-Y. Keung and T.-C. Yuan, *Phenomenology of iquarkonium*, *Nucl. Phys.* **B 811** (2009) 274 [arXiv:0810.1524].

[15] V. Halyo, P. LeGresley and P. Lujan, *Massively Parallel Computing and the Search for Jets and Black Holes at the LHC*, *Nucl. Instrum. Meth.* **A 744** (2014) 54 [arXiv:1309.6275].

[16] V. Halyo, A. Hunt, P. Jindal, P. LeGresley and P. Lujan, *GPU Enhancement of the Trigger to Extend Physics Reach at the LHC*, 2013 *JINST* **1310** P10005 [arXiv:1305.4855].

[17] ATLAS collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, 2008 *JINST* **3** S08003.

[18] CMS collaboration, *The CMS experiment at the CERN LHC*, 2008 *JINST* **3** S08004.

[19] J. Sanders and E. Kandrot, *CUDA By Example*, Addison-Wesley Professional, (2010).

[20] P. Hough, *Machine analysis of Bubble Chamber Pictures*, *Proc. Int. Conf. High Energy Accelerators and Instrumentation* **C 590914** (1959) 554.

[21] P. Hough, *Method and mean for recognizing complex patterns*, United States Patent 3069654, (1962).

[22] R. Gonzalez and R. Woods, *Digital Image Processing*, Prentice Hall, (1993).

2014 JINST 9 P04005

[23] M. Ohlsson, C. Peterson and A.L. Yuille, *Track finding with deformable templates: The Elastic arms approach*, *Comput. Phys. Commun.* **71** (1992) 77.

[24] J. Illingworth and J. Kittler, *The adaptive Hough transform*, *IEEE Trans. on Pattern Analysis and Machine Intelligence PAMI* **9** (1987) 690.

[25] S. Bailey et al., *Rapid 3D track reconstruction with the BABAR trigger upgrade*, *IEEE Trans. Nucl. Sci.* **51** (2004) 2352.

[26] L.M. de A. Filho and J.M. de Seixas, *Combining Hough transform and optimal filtering for efficient cosmic ray detection with a hadronic calorimeter*, PoS(ACAT08)095.

[27] http://www.colfax-intl.com/xeonphi/CX2265i-XP5.html, (retrieved Oct. 17, 2013).

[28] Open MP application program interface, http://openmp.org/, (retrieved Jan. 29, 2014).

[29] http://en.wikipedia.org/wiki/Loop_tiling, (retrieved Jan. 29, 2014).

[30] R. Krishnaiyer et al., *Compiler-based data prefetching and streaming non-temporal store generation for the intel xeon phi coprocessor*, http://hgpu.org/?p=10253, (retrieved Oct. 17, 2013).

[31] http://www.colfax-intl.com/nd/Servers/CXP9100.aspx, (retrieved Oct. 17, 2013).

[32] http://www.amaxit.com/default.asp, (retrieved Oct. 28, 2013).

[33] Parallel Programming and Optimization with Intel Xeon Phi Coprocessors, Colfax International, 2013, http://www.colfax-intl.com/xeonphi/book.html.

[34] N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*, Addison-Wesley Professional, (2013), http://www.cudahandbook.com/.

[35] CMS collaboration, *Tracking and Primary Vertex Results in First 7 TeV Collisions*, CMS Physics Analysis Summary TRK-10-005.

2014 JINST 9 P04005