Preprint typeset in JINST style - HYPER VERSION

Massively Parallel Computing and the Search for Jets and Black Holes at the LHC

V. Halyo*, P. LeGresley, P. Lujan

Princeton University, Princeton, NJ, USA
E-mail: vhalyo@gmail.com

ABSTRACT: Massively parallel computing at the LHC could be the next leap necessary to reach an era of new discoveries at the LHC post the Higgs like discovery. Scientific computing constitutes an essential part of the LHC experiment, including: operation, trigger, LHC computing GRID, simulation, and analysis. The flexibility of the trigger system and the quest for new physics at the LHC suggests the trigger as the first place to integrate Graphics Processing Unit (GPU) or Many Integrated Core (MIC) cards in its server farm. This cutting edge technology provides not only the means to accelerate existing algorithms but also the opportunity to develop new algorithms that select events that could have previously evaded detection. In this article we describe new algorithms to select prompt or non prompt jet and black hole like objects in the silicon tracker.

KEYWORDS: ATLAS; CMS; Level-1 trigger; HLT; Tracker system; Jets; Black holes.

*Corresponding Author

Contents

1. **Introduction** 1

2. **Physics Motivation** 2

3. **Processor Architecture** 2

4. **Hough Transform Algorithm** 5

5. **Jet and Black Hole Detection** 7

6. **Preliminary Results** 8

7. **Summary** 9

1. Introduction

Data analysis at the LHC requires approaches to processing that are effective at finding the underlying physics phenomena while still being cost efficient. Using computing technologies derived from consumer products results in lower hardware acquisition costs because many of the research, development, and manufacturing costs are amortized over much larger volumes. These consumer products are increasingly mobile devices such as phones, tables, and notebook computers so there is significant interest in energy efficiency to maximize battery life. This focus on energy efficiency also has the potential to benefit more traditional areas of computing and data analysis by reducing operating costs such as electrical power and cooling.

Leveraging processors derived from consumer products helps minimize costs associated with purchasing and operating large compute installations such as required for the LHC, but there is also a software aspect that needs to be considered. Moore's Law continues to yield ever more powerful processors but the architecture of processors has changed significantly over the past decade. For many years CPUs were able to run the exact same software faster and faster with each new generation of processor. This changed in the early 2000s as Intel and AMD shifted focus to multicore processors featuring at first two cores, but with some newer processors featuring as many as 16 cores.

Parallel processors, in the form of multicore CPUs and more recently highly programmable Graphics Processing Units (GPUs), may require a significant rethink 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 physics Beyond the Standard Model (BSM) 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.

2. Physics Motivation

Various extensions BSM predict the existence of new, strongly interacting particles that lead to final states with high jet multiplicities. Other exotic models might include boosted jets or long-lived neutral particles decaying at macroscopic distances from the primary vertex [1] in the tracker. The key to observing these events is selecting these jets in real time in the trigger system for archiving and further careful, offline analysis.

Both CMS [2] and ATLAS [3] have a typical trigger system with a hierarchy of multiple levels, ranging from fast and relatively simple criteria implemented entirely in hardware and firmware, to more sophisticated software-based analysis. The primary goal of the software based level is to apply a specific set of physics selection algorithms on the events read out and accept the events with the most interesting physics content. This computationally intensive processing is executed on a farm of commercial CPU processors. The quest for new physics and the flexibility of the trigger system suggests that this computer farm is the natural place to integrate GPU or MIC cards. These cards allow for the development of fast and unique algorithms using massively parallel programming architectures to select events that would indicate physics BSM.

In this article newly developed trigger algorithms are described that would be able to select events that include displaced jets originating from long lived particles, or various topologies with boosted jets that will be more frequent as the center of mass energy of the collision at design luminosity is increased. These new triggers will naturally be sensitive to a larger parameter space, including lower-mass Higgs that decay to displaced heavy flavor hadrons, that could have evaded detection in previous experiments. For example, CMS has performed a search for long-lived neutral particles decaying to displaced leptons [4], but the efficiency for measuring leptons originating from a low-mass Higgs of $M_H = 120$ GeV/c^2 is very low due to the lepton momentum requirements in the trigger, resulting in relatively poor limits for this mass region.

Generalization of the trigger algorithm developed in this article would allow searching for the existence of soft displaced black hole like decay. Namely, high multiplicity democratic decay of soft particles that originate a few centimeters from the interaction point. To conclude, introducing massively parallel programming allows for development of algorithms that are computationally infeasible using CPUs and might provide an invaluable way to test for topological signatures that clearly do not exist in the SM.

3. Processor Architecture

When discussing computer processors it is common to bring up Moore's Law. However, it is important to recognize what Moore's Law actually represents. Moore's Law is an empirical observation made by Intel co-founder Gordon Moore in the 1960s that it was economically feasible to double the number of transistors on a single integrated circuit every 24 months. So although modern usage tends to associate Moore's Law with processor performance it is first and foremost about manufacturing technology and the ability to fabricate ever smaller transistors.

In terms of performance, one of the most obvious benefits of smaller transistors is the ability to run processors at faster clock rates. Over the course of a few years processor clock speeds jumped from tens of MHz to GHz speeds. However, manufacturers were not able to fully decrease operating

voltages in a corresponding way and this led to increases in power density and the inability to dissipate so much power from such a small area. For the first time processors had become power limited.

It is fairly clear how a faster clock speed yields a faster processor. What is a little less obvious is how more transistors can make a single core processor faster even at fixed clock speeds. One option is to perform multiple operations at each clock cycle with parallel execution units. In the late 1990s Intel and AMD added something to their processors that has come to be known as SSE, or Streaming SIMD Extensions, where SIMD refers to a type of parallel architecture as defined by Flynn's taxonomy. In the SIMD (Same Instruction Multiple Data) model one instruction such as multiplication, addition, etc. is applied to multiple data items. This parallelism is typically something to be expressed by the software developer based on the computations to be performed and the capabilities of the hardware. However, most modern compilers can also target the SSE units in a process typically referred to as auto vectorization. The effectiveness of auto vectorization depends on the capabilities of the compiler, the complexity of the code being analyzed, and the skill of the software developer at writing code that can be successfully analyzed by the compiler.

A potential disadvantage of the SSE approach is that it requires software that specifically makes use of it, either through manual programming by the software developer or auto vectorization at compile time. But the additional transistors available at each generation of Moore's Law also allowed hardware designers to implement Instruction Level Parallelism (ILP). Instead of the processor executing one operation per clock cycle (an arithmetic operation, or a load from memory, or a store back to memory, etc.) multiple instructions are executed in parallel. For example, a load from memory is issued for a future arithmetic operation, an arithmetic operation is performed, and a store back to memory of a previous arithmetic operation is issued. In x86 processors this is determined dynamically as the program is executing and allows arbitrary single-threaded software to run faster by exploiting parallelism within the instruction stream.

The implementation of ILP and the associated techniques of instruction pipelining, out-of-order execution, speculative execution, and hardware branch prediction helped consume many of the additional transistors made available through Moore's Law. But in addition to ILP, the growing gap between the performance of the processors running at GHz speeds and the off chip memory systems necessitated adding a complicated cache hierarchy to modern processors.

While compute performance may increase around 40% per year in accordance with Moore's Law, memory performance increases approximately 10% year. If left unchecked this gap in performance would have resulted in processors that had the potential to be very computationally powerful but would be limited in actual performance because of the constant need to wait on the memory system. To address this modern processors have a complicated cache hierarchy not only for data, but also for instructions, and even for translating virtual memory addresses to physical memory addresses in the form of Translation Lookaside Buffers (TLB). Additional logic in the form of hardware prefetchers observes memory access patterns in real time and attempts to prefetch into the caches data and instructions that are likely to be needed in the future. The disparity between processor and memory performance is so bad that the vast majority of the chip area in modern processors is devoted not to computation but to data handling. As shown in Figure 1 the L3 cache plus the memory controller and other I/O portions of a chip can easily represent around 50% of the area. Within each core there is an L1 and L2 cache, plus all of the logic associated with ILP for

that core. The actual chip area devoted to computations is remarkably small.

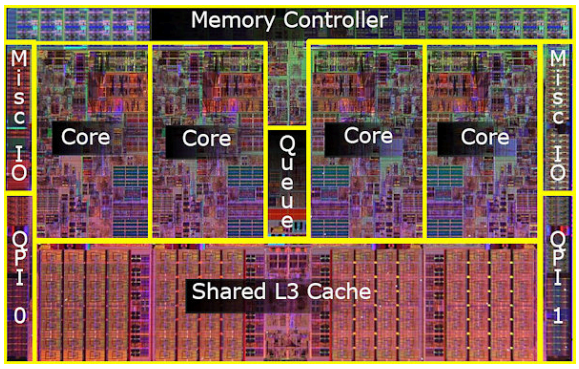


Figure 1. Intel Core i7 die showing major components of the processor (image from Intel).

Faster clock speeds, ILP, and hardware prefetechers had the advantage of using existing software and simply running it faster. However, there are limits to what the hardware can do to automatically extract ILP and predict the memory access pattern of arbitrary applications. Deep instruction pipelines, which in some processors exceeded 30 stages, have issues with hazards that would affect the correctness of results if not correctly handled by the processor, stalls due to unavailable inputs, and mispredicted branching behavior. At some point this reaches a level of diminishing returns where the hardware complexity, and hence cost, outweighs the performance benefits.

With faster clock speeds not feasible and ILP also reaching its limits, hardware designers in the early 2000s turned to multicore to utilize all of those additional transistors available to them from Moore's Law. As seen in Figure 1, the hardware designers develop the design for a core, plus the hardware logic to link them together, and can then place as many cores on a single chip as current manufacturing technology allows. With multiple cores, independent instances of applications (web browser, word processor, anti-virus, etc.) can run in parallel on the multiple cores without application level changes. But if one application such as a video transcoder wants to speed up the processing of a single video it must be specially written to make use of multiple cores.

Modern CPUs with ILP, SIMD units, and multicore are highly parallel processors yet still retain the ability to efficiently run single-threaded, sequential software because so many applications make little to no use of parallel computing. An alternate approach would be to design an inherently parallel processor that is only designed to run parallel programs efficiently. Such a processor would not need to run single-threaded, sequential software and therefore could devote more chip area to computations and less to features like ILP, caches, and hardware prefetechers. An example of a processor that is specialized to running parallel workloads is GPUs.

As implied by the name GPUs were designed as specialized processors for processing graphics. In the 1990s the initial GPUs targeted towards consumer applications handled the computations associated with computing the color of each pixel in a process referred to as pixel shading. Like any integrated circuit, GPUs also take advantage of Moore's Law. The pixel shaders became more programmable so that game designers could create better looking graphics that were not the same as the graphics from everyone else, and additional functionality in the form of vertex shaders were added to the GPU for processing the geometry used in 3D graphics.

CPUs are designed to be multipurpose and run operating systems, word processors, spreadsheets, games, etc. while the GPU was more special purpose and only intended to do graphics computations. Because the GPU was intended for a specialized workload, the designers had a choice when using additional transistors to make the GPU more powerful. One option would be to follow the CPU model and create one or more very complicated cores with ILP, caches, etc. or design a simpler, smaller core that allows many more cores per chip. The disadvantage of simpler, smaller cores is that such a core probably will not be as fast at running single threaded code but that did not matter for the GPU since it targeted a special application area with lots of available parallelism. Modern GPU designs use the simpler, smaller core approach and modern high end GPU hardware has thousands of parallel execution units.

In addition to making GPUs more computationally powerful, the programmability has been improved through both changes in hardware and new software tools. The programmability of GPUs using options such as CUDA, OpenCL, and OpenACC make the hardware usable for many applications beyond graphics, and without requiring programmers to have a specialized graphics background. However, a strong background in parallel computing is still required to make effective use of a GPU.

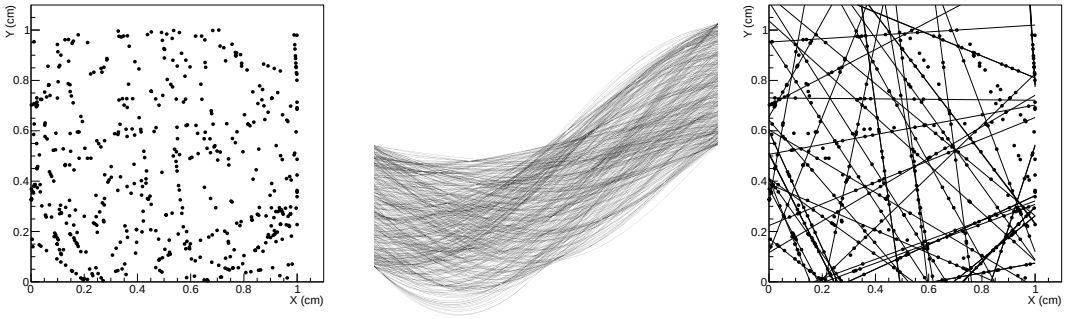
With the growing success of GPUs as a massively parallel processor well suited to more general purpose applications, Intel has introduced a line of products based on a Many Integrated Core (MIC) architecture and marketed as Xeon Phi. The Xeon Phi products physically look similar to a GPU in that they plug into a host system via PCI Express. And from an architecture perspective they also have many similarities to GPUs. Xeon Phi is based on an x86 Pentium core 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 SIMD units known as Advanced Vector Extensions 2 (AVX2) have been added to the core. Because of the simple nature of the core a single Xeon Phi processor may have up to 64 cores.

Regardless of whether the future is GPUs or the architecturally similar Xeon Phi, it seems clear that the future of computing is massively parallel processors. This will require rethinking algorithms and their implementations to make effective use of these highly parallel processors. Algorithms that have been viewed as computationally infeasible should be revisited, and current algorithms that do not parallelize well will need to be reworked to take advantage of modern processors.

4. Hough Transform Algorithm

As an example of implementing a computationally intensive algorithm on a massively parallel processor the authors have previously developed an implementation of the Hough transform using CUDA [5]. The Hough transform 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 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 look-

ing 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.



(a) (b) (c)

Figure 2. Hough transform algorithm applied to a simple example. Left: Hits in a simulated event with 50 straight-line tracks and 10 hits per track. Center: Each hit results in a sinusoidal 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.

One advantage of the Hough transform is that it is tolerant to missing data or data that does not exactly fit the candidate features. This could occur if there is noise in the data, or if the resolution of the data results in features that cannot be perfectly represented with a given computational discretization. The downside of this robustness is that the Hough transform is computationally expensive. However, the implementation using the GPU has demonstrated significant performance advantages compared to a multithreaded CPU implementation. This performance comparison was made using a self developed CPU implementation because typical implementations such as the one in Intel MKL are sequential implementations that do not utilize multicore CPUs. It was felt that the multithreaded CPU implementation was a more fair way to compare performance, but self implementations can sometimes be biased. Therefore, additional CPU performance optimizations have been implemented in the form of using AVX intrinsics and improvements to the OpenMP parallelization.

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 and 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.

As shown in Figure 3 the combination of improvements resulted in approximately a 40% decrease in the runtime for the CPU implementation.

It is worth mentioning that while the use of AVX has the potential for an 8x improvement when using single precision floating point, only improving the performance by 40% was not unexpected for this code. That is because the previous CPU implementation sought to minimize expensive computations by ensuring that redundant computations were moved outside of loops and intermediate results reused where possible. These existing optimizations meant the implementation was limited more by memory access operations rather than the computational rate. It is always use-

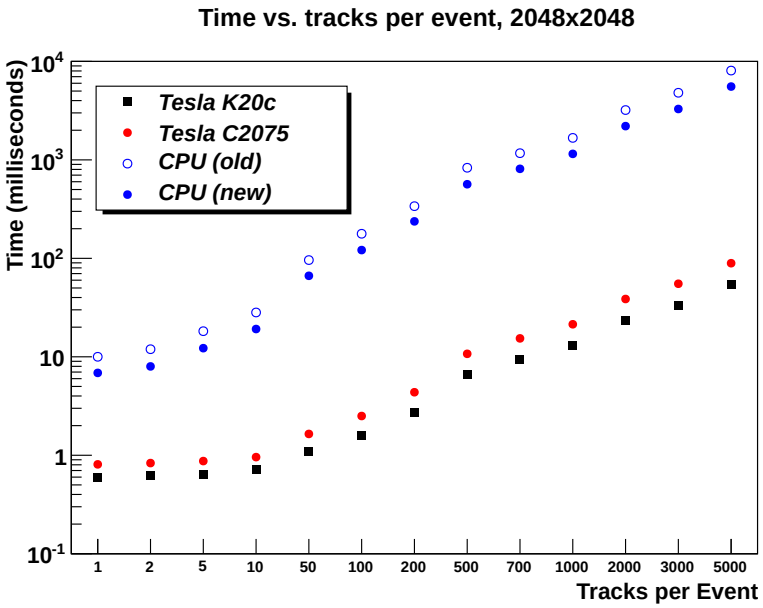


Figure 3. Time performance of old and new CPU implementations compared to NVIDIA Tesla C2075 and K20c. Intel Core i7-3770 CPU running a multithreaded implementation.

ful to understand what aspect(s) of the hardware, such as computations or data access, limit the performance of an algorithm so that time spent on implementing optimizations is used effectively.

5. Jet and Black Hole Detection

The overall data processing pipeline for detection of displaced jets and black holes builds on the existing components of the tracking algorithm as shown in Figure 4. As an initial step the hit data for an event is processed using the Hough transform to compute the parameter space representation. Candidate tracks are identified by finding local maxima, or peaks, in the parameter space. A Kalman filter is then used to further refine the candidate tracks. At this time the Kalman filter is being run using a CPU implementation due to the relatively insignificant amount of time required compared to other aspects of the data analysis. The Kalman filter algorithm is amenable to a GPU implementation which could be undertaken if the computational cost of the Kalman filter becomes a bottleneck.

With candidate tracks identified the corresponding parameter space representation of these tracks is processed by a second application of the Hough transform. In this case the goal is to identify not straight lines but sinusoids which contain a significant number of tracks, indicating that this is a location of a displaced jet or black hole. Again, the location of features of interest correspond to local maxima, or peaks, in the parameter space so the same peak finding algorithm is applied to the parameter space of this second Hough transform. The output of the peak finding can be used directly, as shown here, or if necessary after application of the Kalman filter if needed.

It should be noted that the large number of tracks originating from the interaction point will also be identified as a jet. These false positives can be excluded by filtering out features that are at

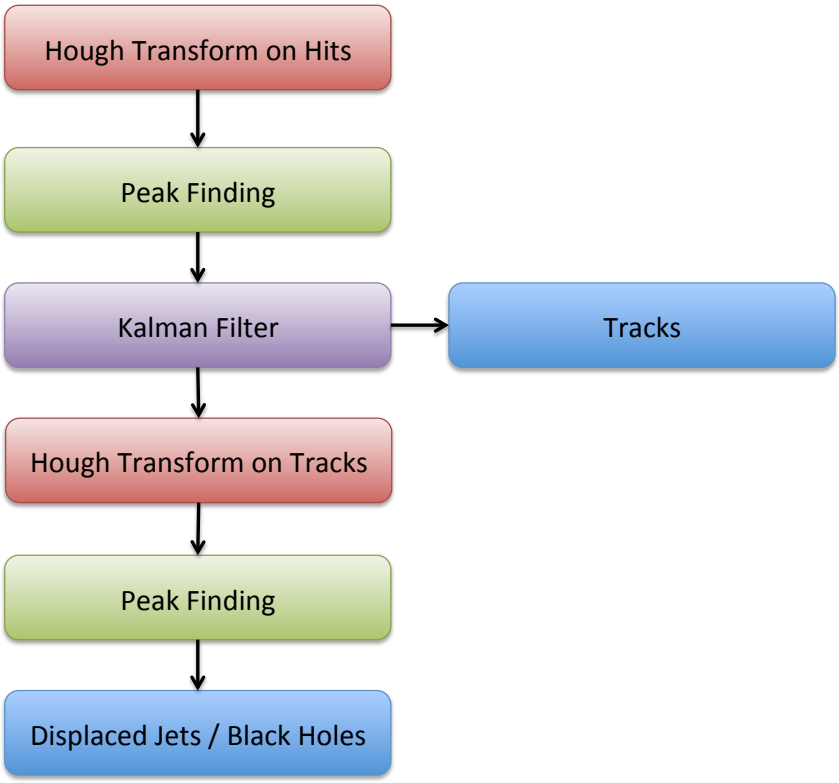


Figure 4. Flowchart showing the steps in processing the input hit data to tracks and displaced jets or black holes.

or extremely close to the known interaction point. This filtering could be done within the Hough transform process itself by setting a lower bound on the radius or at the downstream peak finding step. If a Kalman filter were also incorporated it could also be incorporated at that step as well if desired.

Figure 5 illustrates the detection process for a simple event with four displaced jets. The initial hit data is used to identify tracks as shown on the left in xy space and in the center in parameter space. Applying a second Hough transform to the tracks in parameter space identifies four sinusoids in the second parameter space shown on the right. These four sinusoids correspond to the four vertices of the displaced jets in the original xy space.

6. Preliminary Results

At first glance the proposed enhancement for detecting displaced jets and black holes would seem to double the computational cost due to the need to perform a second expensive Hough transform. However, it is important to note that the first application of the Hough transform and associated peak finding and Kalman filtering results in a significant data dimensionality reduction. Where the first application of the transform has a cost proportional to the number of hits, the second application of the transform has a cost proportional to the number of tracks. For example, with an average

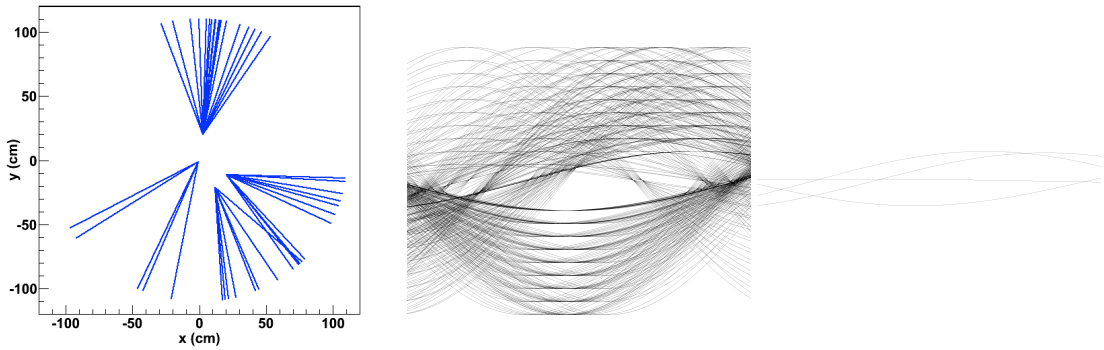


Figure 5. Hough transform algorithm applied to an event with multiple displaced jets. Left: Simulated tracks in the event. Center: The parameter space is difficult to visually interpret. Right: The second Hough transform identifies the sinusoids corresponding to the jet vertices.

of 10 hits per track and in the limit of 100% efficiency and purity the dimensionality of the data is reduced by a factor of 10. Figure 6 shows a time comparison of the baseline tracking algorithm and the enhancement for detecting displaced jets and black holes. Ten percent of the tracks are associated with a displaced jet. So in practice the computational cost increases by approximately 30% compared to the baseline tracking algorithm. This time includes the time for the data transfer and computational time of the Kalman filter running on the CPU and represents an additional opportunity for performance optimization if the Kalman filter were to be moved to the GPU.

In Figure 7 results for the efficiency of detecting a single jet with a varying number of tracks and 10 hits per track are shown. Efficiency is defined as the fraction of simulated jets successfully identified by the algorithm divided by the number of jets known to be present. The efficiency is relatively low for a small number of tracks but quickly rises for 10 tracks and beyond. Because the peak finding has been optimized for track detection where 7 or more hits per track are expected, simple reuse of this code explains the efficiency results. Separate tuning of the peak detection for a fewer number of votes per feature could improve the jet finding efficiency.

Identification of a jet or black hole becomes more difficult if there are multiple such features in the event. Figure 8 shows efficiency with a varying number of jets per event. Each jet has 10 tracks with 10 hits per track and dispersion angles between 15 and 60 degrees. There seems to be a slight trend towards lower efficiency as the number of jets per event is increased.

Finally we consider a more realistic event made up of 3000 total tracks of 10 hits each, with some of the tracks being part of a varying number of jets made up of 10 tracks each and having dispersal angles between 15 and 60 degrees. Because there are now tracks originating at the interaction point, the jet detection filters out any potential vertices within the 3.0 cm beam pipe radius. As shown in Figure 9 the efficiency seems to be independent of the number of displaced jets.

7. Summary

A previous proposal by the authors to enhance the trigger has been extended to detect displaced jets and black holes, which would be used to detect new physics beyond the Standard Model. The

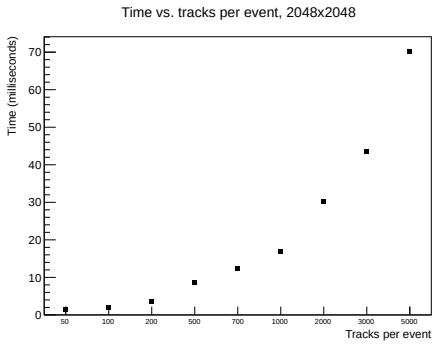


Figure 6. Time performance of the tracking plus displaced jet detection algorithm using an NVIDIA Tesla K20c. Ten percent of the tracks are associated with a displaced jet.

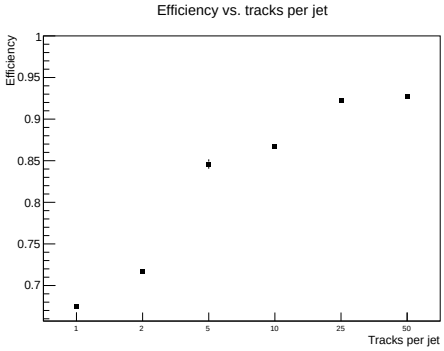


Figure 7. Efficiency of displaced jet detection with varying number of tracks per displaced jet.

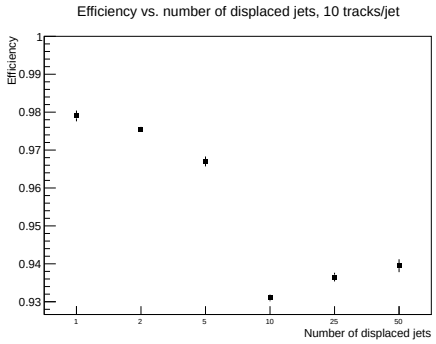


Figure 8. Efficiency of displaced jet detection with varying number of displaced jets per event. Each displaced jet has 10 tracks and a dispersion angle between 15 and 60 degrees.

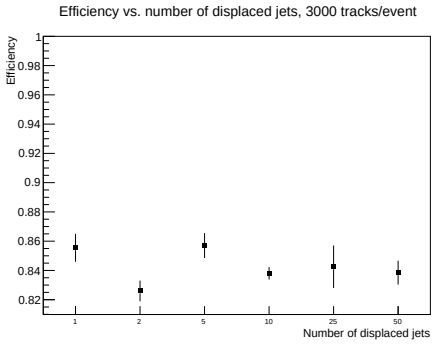


Figure 9. Efficiency of displaced jet detection in the presence of tracks originating from the interaction point. There are a total of 3000 tracks, some of which make up a varying number of displaced jets.

proposed extension uses a second application of the Hough transform to identify tracks originating from a vertex displaced from the interaction point. Although the Hough transform is a computationally expensive algorithm an implementation on a massively parallel processor such as a GPU is significantly faster than implementations on conventional CPUs. Recent work to incorporate additional CPU optimizations yielded approximately a 40% improvement in performance, although the GPU implementation is still approximately 60x faster than the corresponding multithreaded, vectorized CPU implementation. The computational cost increase when performing the second Hough transform is mitigated by the data reduction of the first application of the transform which reduces the data dimensionality from the number of hits to the number of tracks. Benchmarks using data from Monte Carlo simulations showed that the implementation of this proposed trigger enhancement for displaced jets and black holes only increased the runtime by 30%.

Future work will focus on efficiency and purity improvements, as well as investigating further GPU performance optimizations. In addition it would be desirable to compare these performance results with an equivalent implementation on the Intel Xeon Phi. While still based on the x86 archi-

tecture the Phi is designed using much simpler cores, allowing for many more cores per processor. This means that compared to typical multicore CPUs, the Phi is more architecturally similar to GPUs.

References

[1] M. J. Strassler, K. M. Zurek, *Echoes of a hidden valley at hadron colliders*, Phys. Lett. **B**651:374-379,2007

[2] S. Chatrchyan *et al.* [CMS Collaboration], *The CMS experiment at the CERN LHC*, JINST **3**, S08004 (2008).

[3] G. Aad *et al.* [ATLAS Collaboration], *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3**, S08003 (2008).

[4] S. Chatrchyan *et al.* [CMS Collaboration], *Search in leptonic channels for heavy resonances decaying to long-lived neutral particles*, JHEP **1302**, 085 (2013) [arXiv:1211.2472 [hep-ex]].

[5] V. Halyo, A. Hunt, P. Jindal, P. LeGresley, P. Lujan, *GPU Enhancement of the Trigger to Extend Physics Reach at the LHC*, [arXiv:1305.4855 [physics.ins-det]].