

OPEN ACCESS

Many-core applications to online track reconstruction in HEP experiments

To cite this article: S Amerio *et al* 2014 *J. Phys.: Conf. Ser.* **513** 012002

View the [article online](#) for updates and enhancements.

You may also like

- [Parallel track reconstruction in CMS using the cellular automaton approach](#)
D Funke, T Hauth, V Innocente et al.
- [A quantum algorithm for track reconstruction in the LHCb vertex detector](#)
D. Nicotra, M. Lucio Martinez, J.A. de Vries et al.
- [Optimization of the LHCb track reconstruction](#)
Barbara Storaci and on behalf of the LHCb Collaboration



ECS
The
Electrochemical
Society
Advancing solid state &
electrochemical science & technology

DISCOVER
how sustainability
intersects with
electrochemistry & solid
state science research

Many-core applications to online track reconstruction in HEP experiments

S. Amerio¹, D. Bastieri¹, M. Corvo¹, A. Gianelle¹, W. Ketchum²,
T. Liu³, A. Lonardo⁴, D. Lucchesi¹, S. Poprocki⁵, R. Rivera³,
L. Tosoratto⁴, P. Vicini⁴ and P. Wittich⁵,

¹ INFN and University of Padova, Padova, Italy

² Los Alamos National Laboratory, Los Alamos, New Mexico, USA

³ Fermi National Accelerator Laboratory, Batavia, Illinois, USA

⁴ INFN Roma, Rome, Italy

⁵ Cornell University, Ithaca, New York, USA

E-mail: silvia.amerio@pd.infn.it

Abstract. Interest in parallel architectures applied to real time selections is growing in High Energy Physics (HEP) experiments. In this paper we describe performance measurements of Graphic Processing Units (GPUs) and Intel Many Integrated Core architecture (MIC) when applied to a typical HEP online task: the selection of events based on the trajectories of charged particles. We use as benchmark a scaled-up version of the algorithm used at CDF experiment at Tevatron for online track reconstruction - the SVT algorithm - as a realistic test-case for low-latency trigger systems using new computing architectures for LHC experiment. We examine the complexity/performance trade-off in porting existing serial algorithms to many-core devices. Measurements of both data processing and data transfer latency are shown, considering different I/O strategies to/from the parallel devices.

1. Introduction

Real-time event reconstruction plays a fundamental role in High Energy Physics (HEP) experiments at hadron colliders. Reducing the rate of data to be saved on tape from millions to hundreds of events per second is critical. To increase the purity of the collected samples, rate reduction has to be coupled with an initial selection of the most interesting events. In a typical hadron collider experiment, the event rate has to be reduced from tens of MHz to a few kHz. The selection system (trigger) is usually organized in multiple levels, each capable of performing a finer selection on more complex physics objects describing the event. Trigger systems usually comprise a first level based on custom hardware, followed by one or two levels usually based on farms of general purpose processors. At all levels, latency is a concern: for a fixed processing time, the faster a decision is rendered about accepting or rejecting an event improves the purity of the collected data sample. The possibility of using commercial devices at a low trigger level is very appealing: they are subject to continuous performance improvements driven by the consumer market, are less expensive than dedicated hardware, and are easier to support. Among the commercial devices, many-core architectures such as Graphic Processing Units (GPUs) [1] and Intel Many Integrated Core (MIC) [2] are of particular interest for online selections given their great computing power: the latest NVIDIA [3] GPU architecture, Kepler,



exceeds Teraflop computing power. Moreover, high-level programming architectures based on C/C++ such as CUDA [4] and OPENCL [5] make programming these devices more accessible to the general physicist user. The goal of this study is to investigate the strengths and weaknesses of many-core devices when applied in a low latency environment, with particular emphasis on the data transfer latency to/from the device and the algorithm latency for processing on the device in a manner similar to a typical HEP trigger application, and to understand the cost/complexity ratio of porting legacy serial code to many-core devices.

We showed initial studies on GPU performance in low-latency environments ($\approx 100 \mu s$) in previous papers [6, 7]. In this paper we extend those studies to include other many-core architectures (Intel MIC in addition to GPUs). The algorithm run on the parallel architecture is a complete version of the fast track-fitting algorithm of the Silicon Vertex Tracker (SVT) system at CDF [8]. Starting with a serial algorithm implemented on a CPU, we test an *embarrassingly parallel* algorithm on the Intel MIC environment. In this case each event is handled independently by a core on the accelerator, and the parallelization is achieved with only minor changes to the legacy code and is only possible in the Intel MIC environment. Next we consider an algorithm where we unroll three internal nested loops and run these in parallel on a GPU, using the CUDA environment. This second approach is programmatically more complicated and less trivial to implement. In neither case have we re-thought the basic algorithms or the data structures used. To achieve optimal performance, these steps would have to be taken. As one might expect, the improvement from the first approach is rather modest, and the second approach shows larger performance gains. For GPUs, we also test different strategies to transfer data to and from the device.

2. SVT track fitting algorithm

The Silicon Vertex Trigger (SVT) [8, 9] is a track reconstruction processor used in the CDF experiment at Tevatron accelerator. It reconstructs tracks in about $20 \mu s$ in two steps: first, low resolution tracks (*roads*) are found in each event among the energy deposits left in the tracking detector by charged particles; second, track fitting is performed on all possible combinations of hits inside a road. This algorithm uses a linearized approximation to track-fitting as implemented in hardware (described in greater detail in [10]). With the linearized track fit of the SVT approach, the determination of the track parameters (p_i) is reduced to a simple scalar product:

$$p_i = \vec{f}_i \cdot \vec{x}_i + q_i,$$

where \vec{x}_i are input silicon hits, and \vec{f}_i and q_i are pre-defined constant sets. For each set of hits, the algorithm computes the impact parameter d_0 , the azimuthal angle ϕ , the transverse momentum p_T , and the χ^2 of the fitted track by using simple operations such as memory lookup and integer addition and multiplication.

We ported the track fitting as it is well suited to parallelization - each track can be handled independently.

2.1. Code implementation

The starting point of our studies is the SVT track fitting simulation code, written in the C language. SVT track fitting is divided into three main functions: first, the unpacking of input data and filling of all the necessary data structures; second, the computation of all possible combinations of hits in each road and third, the linearized track fit of each combination of hits. Three main loops are present - on events, roads and hit combinations. To be run on GPUs, the code has been ported to CUDA: each step - unpack, combine and track fit - is performed by a specific kernel; the three nested loops are unrolled so that each GPU thread processes a single combination of hits. To run on MIC, where cores are more powerful but fewer in number,

Model	Tesla M2050	Tesla K20m	GeForce GTX Titan	MIC 5110P
Performance (SP, GFlops)	1030	3520	4500	2022
Memory bandwidth (GB/s)	148	208	288	320
Memory size (GB)	3	5	6	8
Number of cores	448	2496	2688	240
Clock speed (GHz)	1.15	0.706	0.837	1.053

Table 1. Capabilities of the many-core devices used in this study, according to the manufacturer’s specifications. The first three are NVIDIA GPUs and the final is an Intel Xeon Phi.

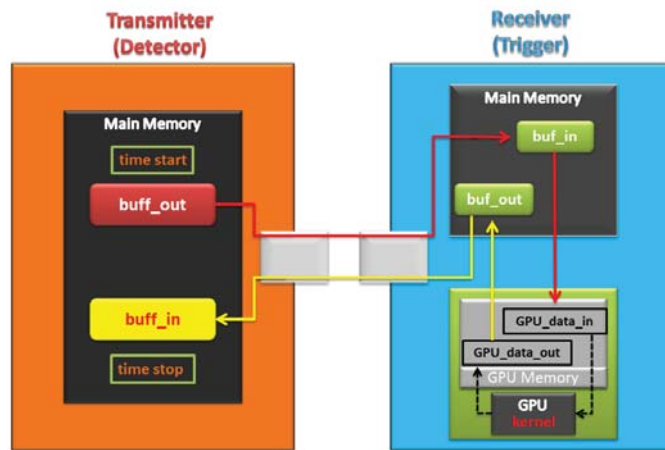


Figure 1. Data flow. Data is sent from the transmitter PC to the receiver PC, where it is processed by the GPU before being returned to the transmitter PC. The transmitter plays the role of the detector as the source of the data and as an upstream trigger processor as the data’s ultimate sink. The receiver PC plays the role of a component in the trigger system.

we adopted the so-called *embarrassingly parallel* approach and used PRAGMA OPENMP [11] for statements to unroll only the external loop on the events, so that each core processes a single event: the porting requires much less effort compared to CUDA, but the level of parallelism is limited.

3. Experimental setup and data flow

The many-core devices used in this study are listed in Table 1. The GPUs include a less expensive gaming class GPU (GTX) and ones optimized for scientific computing (Tesla). The MIC corresponds to a Xeon Phi introduced in November 2012.

To measure the data transfer latency we use a computing cluster composed of 12 identical nodes. Each node contains a Intel Xeon E6520 2.4 GHz CPU and two Tesla M2075 GPU cards. The nodes are connected by InfiniBand communication links using Connect-X2 Mellanox or APEnet+ adapters. APEnet+ is an FPGA-based PCIe board supporting peer-to-peer communication with Tesla and Kepler cards [12]. Two nodes of this cluster are used to measure data transfer latency, one acting as a transmitter and the other as a receiver. Data are transferred from the transmitter to the receiver, processed on the GPU and sent back to the receiver (see Fig. 1). The latency for a complete loop is measured on the transmitter using standard C libraries. In this setup, the transmitter can represent the detector, as the source of the data, or an upstream trigger processor, as the ultimate sink of the data, while the the receiver is the

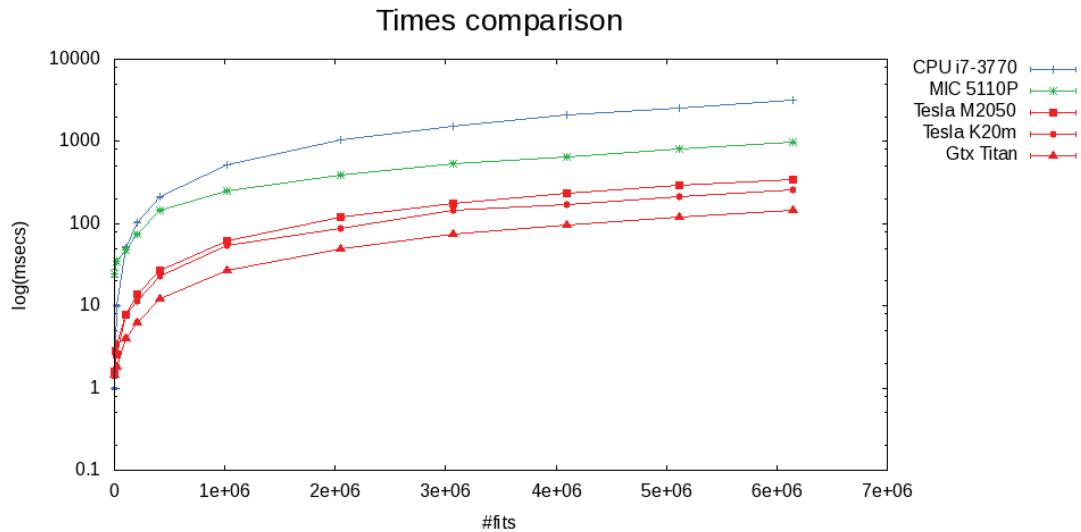


Figure 2. Algorithm-only comparison for timing as a function of the number of track fits. We compare timing on CPUs (serial), Intel MIC (embarrassingly parallel), and GPUs (fully parallel), in blue, green, and red, respectively. The GPUs exhibit the best performance due to the full parallelization.

trigger system: the time to transfer data to the receiver is thus a rough estimate of the latency to transfer the data from the detector front-end to the trigger system.

4. Results

The input data consists of events with a fixed number of roads and combinations: each event has 2048 combinations to be fitted. To explore different data-taking conditions, the number of events ranges from one to 3000, *i.e.*, from 2048 to about six millions of combinations to fit.

4.1. Data processing

Each data sample is processed 100 times by the track fitting algorithm. The average latency as a function of the number of fits is presented in Fig. 2 for the serial, embarrassingly parallel and parallel algorithms. We see that the embarrassingly parallel algorithm gives a modest increase with respect to the serial (CPU) algorithm. Switching to a fully parallel algorithm affords a much more significant speed improvement. Accelerators performance drop with decreasing number of fits, as can be seen in Fig. 3. Figure 4 shows the speed-up with respect to the serial algorithm run on a standard CPU (Intel Xeon E5630): the maximum gain is obtained processing at least 500 events. This means that to fully exploit parallel architectures millions of fits have to be performed in parallel.

4.1.1. Breakdown of computing time In Fig. 5 we show the fractional time spent in various parts of the algorithm for the embarrassingly parallel algorithm (on Intel MIC) and the parallel algorithm (on NVIDIA Titan GPU), as a function of the number of fits. On both accelerator cards the fractional times are constant for more than 500 input events, where computing resources are saturated. Unlike the MIC, the fit stage takes most of the time on the GPU: this could be caused by the intense memory access frequency intrinsic to this part of the algorithm.

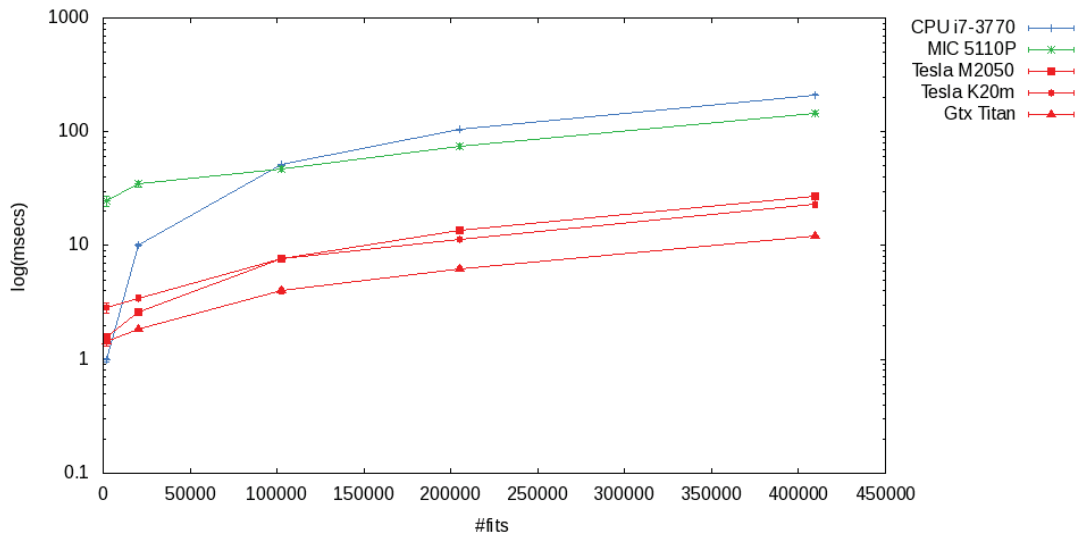


Figure 3. Algorithm-only comparison for timing as a function of the number of track fits: zoom in the low number of fits region. At low number of fits, the CPU performs better, due to start-up costs associated with data transfers to the accelerator card.

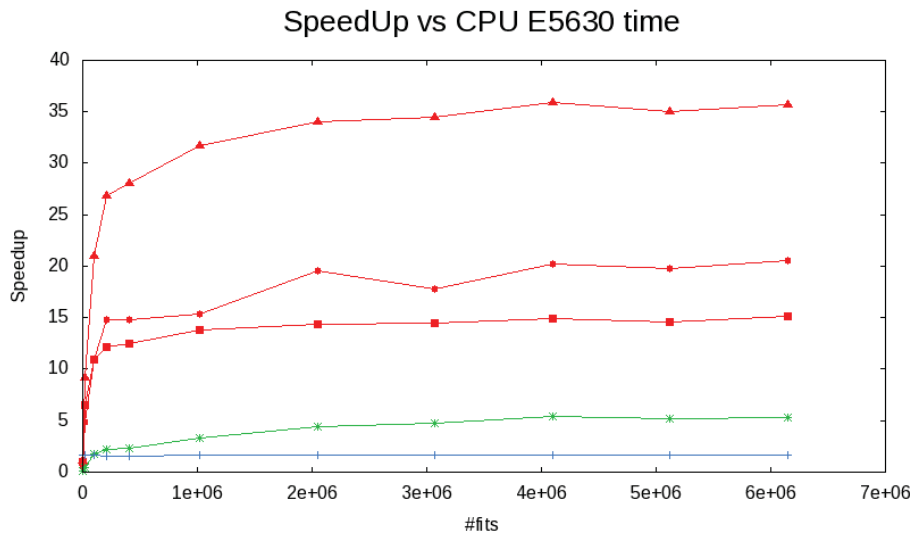


Figure 4. Speed-up with respect to a standard CPU (Intel Xeon E5630). The speed-ups plateau after about two million fits.

4.2. Data transfer

The experimental setup described in Fig. 1 allows us to test different data transfer strategies to the GPU. The standard data transfer strategy is via the system memory, where the PCIe adapter card and the GPU allocate *separate* buffers on the system memory for the copy (as shown in Fig. 6(a)). This is inefficient, as the data are copied twice in the system memory before being transferred to the GPU/PCIe card. Data may also be transferred using Direct Memory Access (DMA, GPUDirect [13]) to the CPU memory: the PCIe card and the GPU share the *same* buffer on the CPU memory; as a result the data are copied only once in the CPU memory (Fig. 6(b)). With our experimental setup two additional copy strategies can be

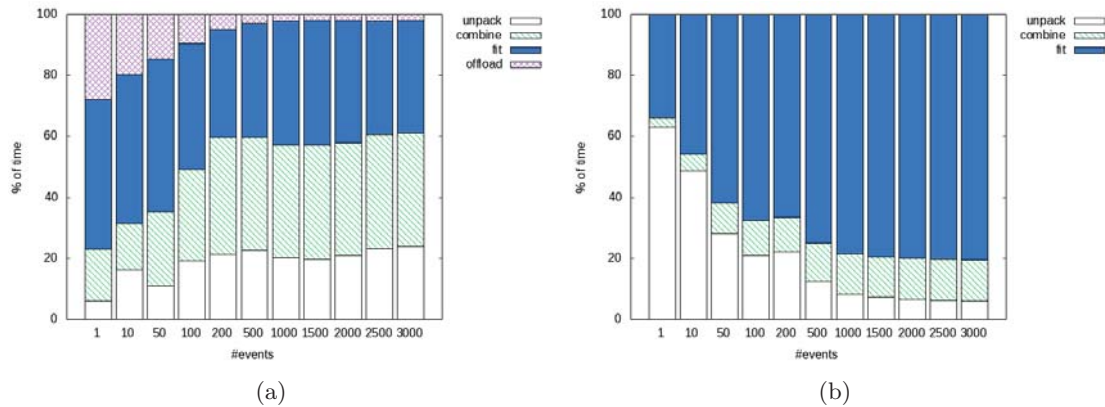


Figure 5. Breakdown of computing time for MIC (a) and the GTX Titan GPU (b). White corresponds to unpacking, green hash corresponds to generating hit combinations, solid blue is the linearized track fit, and magenta cross-hatch corresponds to offloading (MIC only). For MIC, combinations and fitting take the same amount of time for large number of events. For GPU, fitting dominates.

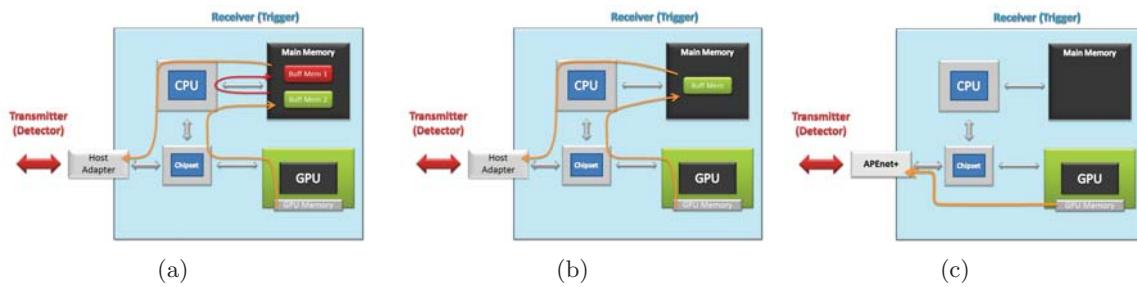


Figure 6. Standard data transfer (a), via GPUDirect (b) and via GPUDirect with P2P support (c). In (a), two buffers are required in the main memory. In GPUDirect (b), one of the main memory buffers is eliminated. In GPUDirect with P2P support, data is sent directly from the APEnet+ transceiver to the GPU memory.

tested which are the results of different levels of optimization of the GPUDirect protocol:

- CUDA-Aware MPI, where the copy latency is further reduced by automatically allocating the buffer on the CPU memory;
- peer-to-peer (P2P) strategy, when data are transferred directly to the GPU, without any intermediate copy to the CPU (Fig. 6(c)).

In Fig. 7 we show the total latency (data transfer, copy to/from the GPU and data processing on the GPU) as a function of data packet size when data are transferred using GPUDirect v1.0, CUDA-aware MPI and P2P. For the packet sizes considered in this test CUDA-aware MPI gives the best performance. This is expected as P2P is optimized for small packet sizes (see also [7] and [14]). As a matter of fact, for larger packet size, the channel throughput becomes dominant: the shortest transfer time of CUDA aware-MPI system is easily explained comparing the link bandwidth of Mellanox board (40 Gb/s) with the smaller throughput of a APEnet+ single link (30 Gb/s). The data transfer latency accounts for a significant part of the total latency, as can be seen in Fig. 8: about 20-25% of total latency is due to moving the data to and from the GPU.

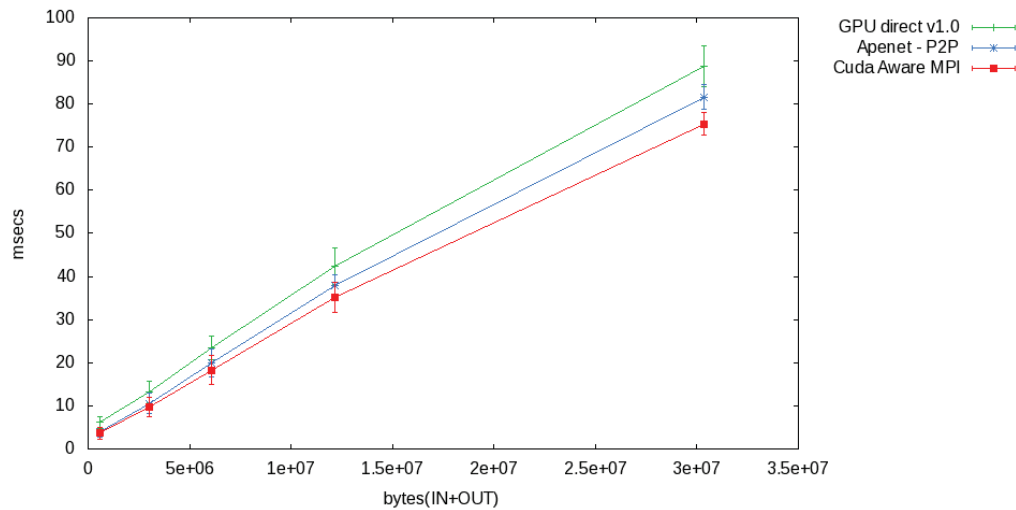


Figure 7. Total latency (data transfer, copy to/from the GPU and data processing on the GPU) as a function of data buffer sizes, for three different levels of optimization of GPUDirect: v1.0, CUDA-aware MPI and P2P. The smallest transferred data packed is 600 kB. CUDA-aware MPI shows the best performance for larger packet size.

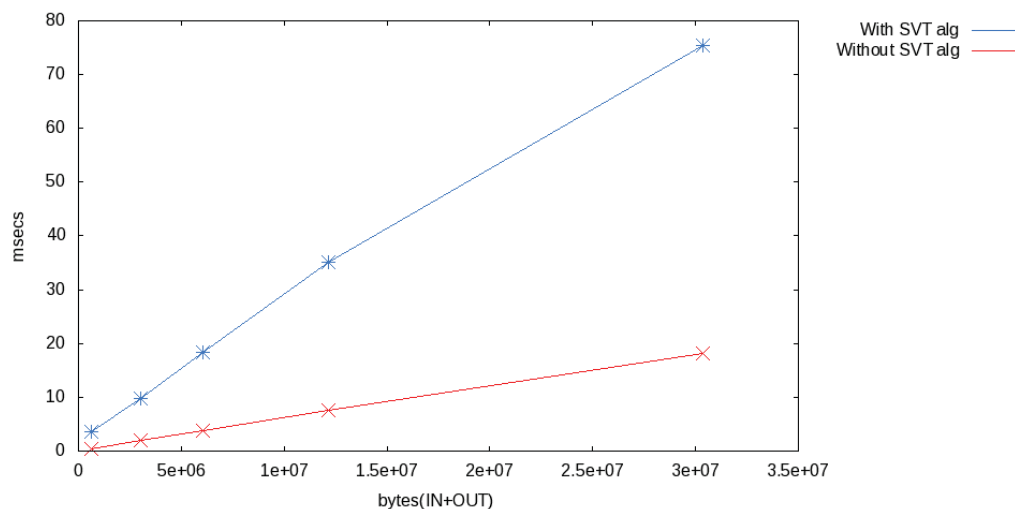


Figure 8. Time per fit, in msec. The two curves show total timing with and without calculations performed on the GPU, thereby showing the considerable time spent in data transfer. About 20-25% of the time is spent in data transfer.

5. Conclusions

We have implemented a full version of the CDF SVT tracking algorithm on GPUs and Intel MIC. We examined a staged approach to using accelerator cards in a hadron collider trigger application. We have demonstrated that in this application, significant gains can be achieved with the ‘embarrassingly parallel’ approach on an Intel MIC architecture, with the smallest amount of required changes to an existing serial code base. However, better performance is achieved with GPUs and a more complete event-level parallelization using CUDA. We have updated latency studies and shown that for larger packet sizes (greater than 600 kB), CUDA-aware MPI outperforms P2P. This result fully agrees with expectations since the P2P mechanism

is optimized (and effective) to reduce the transfer latency for small size packets. Even at large packet size, the data transfer takes an appreciable fraction of the total algorithm time (about 20-25%).

Acknowledgments

The authors would like to thank the Fermilab staff, the FTK group at the University of Chicago and the INFN-APE group in Rome for their support. This work was supported by the U.S. Department of Energy, the U.S. National Science Foundation and the Italian Istituto Nazionale di Fisica Nucleare. This work was partially supported by the EU Framework Programme 7 project EURETILE under grant number 247846.

References

- [1] NVidia Corporation URL <http://www.nvidia.com/object/gpu.html> 1
- [2] Intel Corporation URL <http://tinyurl.com/dxhjs3g> 1
- [3] NVidia Corporation URL <http://www.nvidia.com> 1
- [4] NVidia Corporation URL http://www.nvidia.com/object/cuda_home_new.html 2
- [5] Khronos Group URL <http://www.khronos.org/opencv/> 2
- [6] Ketchum W, Amerio S, Bastieri D, Baucé M, Catastini P, Gelain S, Hahn K, Kim Y, Liu T, Lucchesi D and Urso G 2012 *Proceedings of the 2nd International Conference on Technology and Instrumentation in Particle Physics (TIPP 2011)* vol 37 p 1965 URL <http://www.sciencedirect.com/science/article/pii/S1875389212019207> 2
- [7] Amerio S, Ammendola R, Biagioni A, Bastieri D, Benjamin D, Frezza O, Gelain S, Ketchum W, Kim Y, Cicero F, Lonardo A, Liu T, Lucchesi D, Paolucci P, Poprocki S, Rossetti D, Simula F, Tosoratto L, Urso G, Vicini P and Wittich P 2012 *Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2012 IEEE* p 1806 URL <http://dx.doi.org/10.1109/NSSMIC.2012.6551422> 2, 6
- [8] Ashmanskas W A, Barchiesi A, Bardi A, Bari M, Baumgart M, Belforte S, Berryhill J, Bogdan M, Carosi R, Cerri A, Chlachidze G, Culbertson R, Dell'Orso M, Donati S, Fiori I, Frisch H, Galeotti S, Giannetti P, Glagolev V, Leger A, Liu Y, Maruyama T, Meschi E, Moneta L, Morsani F, Nakaya T, Punzi G, Rescigno M, Ristori L, Sanders H, Sarkar S, Semenov A, Shochet M, Speer T, Spinella F, Vataga H, Wu X, Yang U, Zanello L and Zanetti A 2004 *Frontier Detectors for Frontier Physics: Proceedings* vol 518 p 532 URL <http://www.sciencedirect.com/science/article/pii/S0168900203029450> 2
- [9] Adelman J, Annovi A, Aoki M, Bardi A, Bari M, Bellinger J, Bitossi M, Bogdan M, Carosi R, Catastini P, Cerri A, Chappa S, Dell'Orso M, Ruzza B D, Furi I K, Gianetti P, Giovacchini P, Liu T, Maruyama T, Pedron I, Piendibene M, Pitkanen M, Riesert B, Rescigno M, Ristori L, Sanders H, Sartori L, Shochet M, Simoni B, Spinella F, Torre S, Tripiccone R, Tang F, Yang U and Zanetti A 2007 *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **572** 361 ISSN 0168-9002 frontier Detectors for Frontier Physics: Proceedings of the 10th Pisa Meeting on Advanced Detectors URL <http://www.sciencedirect.com/science/article/pii/S0168900206021498> 2
- [10] Amerio S, Annovi A, Bettini M, Bucciantonio M, Catastini P, Crescioli F, Dell'Orso M, Giannetti P, Lucchesi D, Nicoletto M, Piendibene M and Volpi G 2010 *Journal of Physics: Conference Series* **219** 022001 URL <http://stacks.iop.org/1742-6596/219/i=2/a=022001> 2
- [11] The OpenMP API specification for parallel programming URL <http://openmp.org/wp/> 3
- [12] Ammendola R, Biagioni A, Frezza O, Cicero F L, Lonardo A, Paolucci P, Petronzio R, Rossetti D, Salamon A, Salina G, Simula F, Tantalò N, Tosoratto L and Vicini P 2010 *PoS(Lattice 2012)* 022 URL http://pos.sissa.it/archive/conferences/139/045/Lattice%202011_045.pdf 3
- [13] NVidia Corporation URL <https://developer.nvidia.com/gpudirect> 5
- [14] MVAPICH group see e.g. page 47ff. URL http://www.hpcadvisorycouncil.com/events/2013/European-Workshop/presentations/9_OSU.pdf 6