ALICE TPC Online Tracking on GPU

D. Rohr, K. Aamodt, T. Alt, H. Appelshäuser, A. Arend, B. Becker, S. Böttger, T. Breitner, H. Büsching, S. Chattopadhyay, J. Cleymans, I. Das,Ø. Djuvsland, H. Erdal, R. Fearick, Ø. S. Haaland, P. T. Hille, S. Kalcher,K. Kanaki, U. Kebschull, I. Kisel, M. Kretz, C. Lara, S. Lindal, V. Lindenstruth, A. A. Masoodi, G. Øvrebekk, R. Panse, J. Peschek, M. Ploskon,M. Richter, S. Gorbunov, D. Röhrich, B. Skaali, T. Steinbeck, A. Szostak, J. Thäder,T. Tveter, K. Ullaland, Z. Vilakazi, R. Weis, and P. Zelnicek for the ALICE collaboration

***Abstract*—For online analysis in the ALICE HLT a new, fast TPC tracker was developed. This tracker was adapted to run on graphics cards using the NVIDIA CUDA framework. As the former tracker was already well able to deal with proton-proton events, the adaptation was primarily necessary for heavy-ion events the previous tracker was not able to handle efficiently. The implementation of the algorithm had to be adjusted at many points to allow for an efficient usage of the GPU. In particular, achieving a good overall workload for many processor cores, efficient transfer to and from the GPU, as well as optimized utilization of the different memories the GPU offers turned out to be critical. To cope with these problems a dynamic scheduler was introduced, which redistributes the workload among the processor cores. Additionally a pipeline was implemented so that the tracking on the GPU, the initialization and the output processed by the CPU, as well as the DMA transfer can overlap. The GPU tracking algorithm easily outperforms the CPU version for large events while it entirely maintains its efficiency.**

I. Introduction

M. Richter, Ø. Djuvsland, H. Erdal, K. Kanaki, G. Øvrebekk, Ø. S. Haaland, D. Röhrich, A. Szostak and K. Ullaland are with the Department of Physics and Technology, University of Bergen, Norway e-mail: Matthias.Richter@ift.uib.no

T. Alt, S. Gorbunov, S. Kalcher, V. Lindenstruth, T. M. Steinbeck, and J. Thäder are with the Frankfurt Institut für Informatik IfI, Frankfurt Institut fr Advanced Studies FIAS, Ruth-Moufang-Str. 1, 60438 Frankfurt, Germany.

S. Böttger, T. Breitner, U. Kebschull, M. Kretz, C. Lara, R. Panse, J. Peschek, D. Rohr, R. Weis, and P. Zelnicek are with the Kirchhoff Institute of Physics, University of Heidelberg, Germany

K. Aamodt, S. Lindal, B. Skaali, T. Tveter andare with Department of Physics, University of Oslo, Norway

M. Ploskon is with the Institut für Kernphysik, University of Frankfurt, Frankfurt am Main, Germany

H. Appelshäuser, A. Arend, H. Büsching are with the Institut fr Kernphysik, Max-von-Laue-Str. 1, Goethe-Universitt Frankfurt, 60438 Frankfurt am Main, Germany.

B. Becker, and Z. Vilakazi are with the iThemba LABS, University of Cape Town,PO Box 722, Somerset West 7129, South Africa.

S. Chattopadhyay and I. Das are with the High Energy Physics Division, Saha Institute of Nuclear Physics, 1/AF, Bidhan Nagar, Kolkatta 700 064, India.

J. Cleymans and R. Fearick are with the Department of Physics, University of Cape Town, Private bag X3, Rondebosch 7700, South Africa.

M. Ploskon is with the Physics Division, Lawrence Berkeley National Laboratory,1 Cyclotron Road, 50R4049, Berkeley, CA 94720-8153, United States.

A. A. Masoodi is with the Department of Physics, Aligarh Muslim University, Aligarh 202001, U.P., India.

P. T. Hille is with the Department of Physics, Yale University, New Haven, CT 06520, United States.

I. Kisel is with the GSI Helmholtzzentrum fr Schwerionenforschung GmbH, Planckstr. 1, 64291 Darmstadt, Germany.

THE ALICE High-Level Trigger [1], [2] currently processes proton-proton collisions at 1kHz. It is scheduled that in the end of 2010 the first heavy ion collisions will be processed. The increased complexity of heavy ion collisions makes the tracking computationally more expensive. The Time Projection Chamber (TPC) detector is the main tracking detector of the ALICE experiment, and computing the TPC tracks is one major part of event reconstruction.

In recent years the increase in processor clock speed stagnated but instead a trend to multi- and many-core chips came up. It is obvious, that for raw computation power, the best approach is a big set of small and simple cores as it has been realized within graphics cards for many years now. While at first they could only be used for very special problems using algorithms that had to be developed with a particular architecture in mind, today there are frameworks available to run general purpose code written in high level languages on GPUs with little changes.

The CA Tracking algorithm used in the Alice HLT for TPC online tracking has been developed by Sergey Gorbunov [3] with multi-core support in mind. It includes combinatorics based on the Cellular Automaton principle to determine track candidates in combination with a Kalman filter [6] step for the track fit. All steps can easily be spread on many independent processors. Proton-Proton collisions resulting in up to several hundreds of clusters can already be handled by the HLT compute farm. Primarily targeted at processing upcoming Pb-Pb events with, in the worst case more than 10.000 tracks and several million clusters (see Fig. 1 and Fig. 2), the tracker was adjusted to run on GPUs. A framework being able to run the same source code on CPU as well as GPU was developed, where the same source files are included in wrappers for both processor types. This assures that code maintainability does not suffer.

II. Tracking on GPU

During the tracking there are 5 steps with non negligible requirement of computation time: Initialization, Neighbours Finding, Tracklet Construction, Tracklet Selection and Tracklet Output. Out of these the Tracklet Construction contains all the mathematics and most non trivial calculations while consuming 50% of the time. It is therefore both, the part best suited for running on a GPU and the part with most sense in optimizing it. Currently The hardware equipped in the HLT are

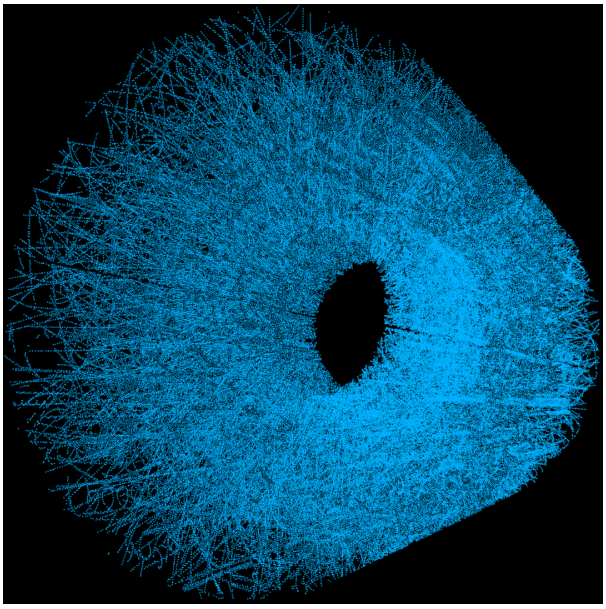

Fig. 1. Clusters of (non central) Pb-Pb event

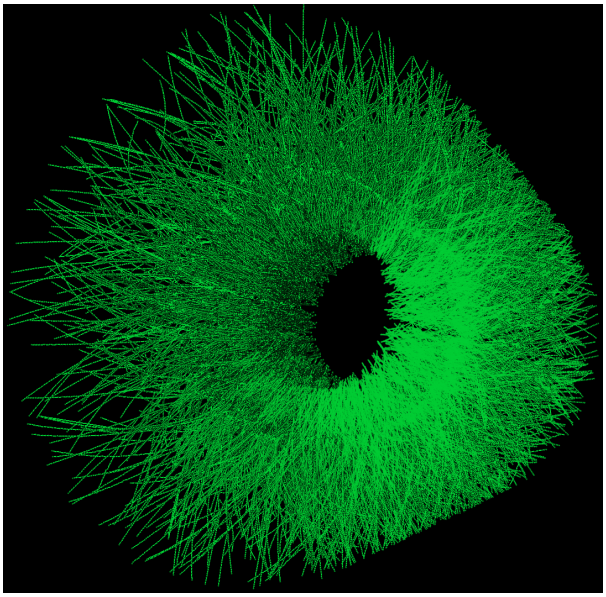

Fig. 2. Tracks found by GPU tracker in event

GT200 chips from NVIDIA. All above-mentioned steps have been ported to CUDA with the most effort put into the Tracklet Construction. The GPU tracker is implemented in a way, that the main tracking algorithm is contained in common source code for the CPU and GPU versions. Only two specialized wrappers are used for both architectures. The common source file is included in both wrappers and processed by the CPU and GPU compiler. This way changes to the algorithm have to be applied only once.

Since current GPU chips show good performance only for single precision calculations, the whole tracker code uses single precision only. An adaptation of the Kalman filter assures numerical stability to the algorithm in single precision [5], [4].

To efficiently run the Tracklet Construction on the GPU, a basic understanding of the GPU's architecture is needed. The GT200 chip consists of 30 independent multiprocessors with 8 ALUs each. Each multiprocessor can handle a vast number of threads in parallel, depending on the algorithm itself one should have about 256 concurrent threads running on each multiprocessor for fully exploiting the GPU. The threads running on a multiprocessor are organized in warps of 32 threads each. All threads in one warp can only execute one particular common instruction. If different threads are to execute different instructions, for example due to branching in the code, these operations have to be serialized.

III. Optimizations for GPU tracking

The GPU implementation of the Tracklet Construction has each tracklet processed by a different thread. The problem arising here is caused by different lengths of the tracklets. As a matter of fact all threads within one warp have to wait for the one thread processing the longest tracklet, even if their current task is already finished. This resulted in the GPU Utilization staying below 20% for the first implementation (See Fig. 3). This was solved by introducing a custom scheduler. One thread only extrapolates a tracklet for a constant number of rows. Afterwards all unfinished tracklets are redistributed among threads and even multiprocessors. Further some pre-filtering was introduced to remove very short tracklets from the queue before even starting the extrapolation. For the scheduler to work efficiently the tracker is able to process multiple slices in parallel. This ensures that there are always enough threads available for scheduling. By applying these changes the GPU utilization raised to almost 70% (Fig. 4). Additionally the memory layout was changed in such a way, that threads which are executed in parallel access consecutive memory addresses. This is done by interleaving the data structures for different threads. The GPU memory controller can coalesce accesses from different threads into one single memory transaction.

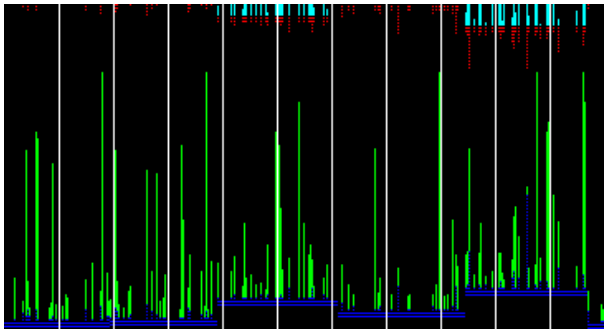

Fig. 3. GPU utilization without scheduling during Tracklet Construction [1]

Apart from the Tracklet Construction also the Neighbours Finding and Tracklet Selection were ported on the GPU. The performance of the neighbours finder could be significantly

[1]White borders seperate threads of one warp. Colors stand for: black: idling, colored: different states during Tracklet Construction.

[2]The three rightmost threads belong to different multiprocessors and are scheduled separately.

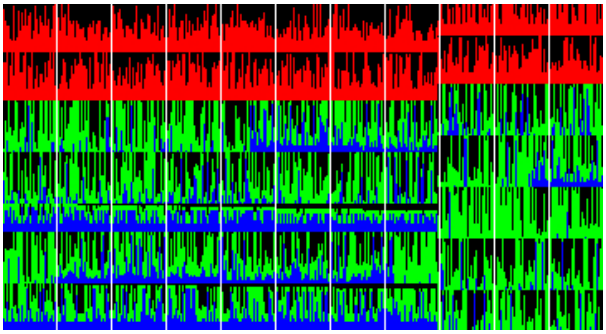

Fig. 4. GPU utilization with scheduling during Tracklet Construction [1] [2]

improved by caching intermediate data in the fast shared memory of the NVIDIA GPU. However the current memory is insufficient for all intermediate data. Fortunately the memory size will be increased with the next GPU generation possibly resulting in a further speedup.

Running the Tracklet Selection on the GPU is necessary because even though it is slower compared to the CPU it greatly decreases the output of the pipeline and thus the amount of data that is transferred back to the host. Contrary to all these tasks, the Initialization and Output steps do not involve computation but instead have lots of random memory access reading requiring most data only once. This is not well suited for a GPU, especially considering the additional data transfer required, but can benefit from big and advanced caches of state of the art CPUs and therefore should stay on the CPU.

Keeping the GPU cores operating at full capacity is the main objective. Since multiple slices are handled simultaneously anyway, to allow efficient scheduling, the steps are pipelined asynchronously using both, CPU and GPU while data is transferred via DMA. This way, after having initialized the tracker data structures for the first slice, the CPU can immediately process the next slice while the GPU starts tracking the first one, as can be seen in Fig. 5.

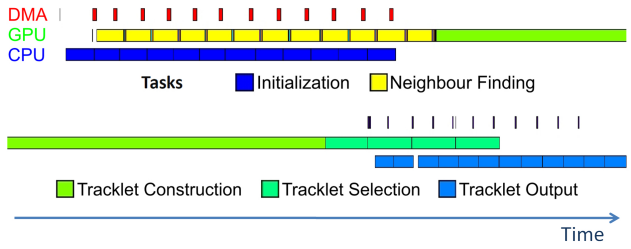

Fig. 5. Asynchronous event processing

IV. Integration in the HLT framework

The general Tracker interface that was developed allows for using both the CPU and GPU tracker within different frameworks. The first and most important one to mention is the HLT framework. Besides that it is possible to run the GPU tracker from AliRoot which is the ALICE offline framework for data analysis, event reconstruction, and simulation. Both build systems needed to be altered in order to invoke the CUDA compilation toolkit. In addition a standalone version was developed and is maintained for debugging and profiling reasons. The HLT uses an AFS file system to store the libraries for all nodes on a common storage. The CUDA framework is not installed on all nodes of the cluster, however, the CUDA runtime library does not support late binding. Therefore libraries referencing the CUDA runtime can only be used on nodes with the CUDA framework installed. To account for this all GPU related code is separated from the rest of the framework in a separate library. This library offers a plain C interface for the creation and destruction of GPU tracker objects that can be used from the framework because of the abstract interface. Another problem that arose is that the CUDA context is thread local, while the HLT framework may uphold multiple threads, all allowed to access the GPU tracker object. Additionally the ROOT C-interpreter (CINT) is unable to process CUDA files. However this is required in order for the logging system to work because it is based on ROOT macros. This is solved by making CINT interprete a fake file stripped of all explicit GPU content.

V. Efficiency

In the GPU Tracking algorithm the order in which tracks are reconstructed is not deterministic. Clusters are assigned to tracks according to several criteria. However, if none of them apply clusters are assigned according to the first-come-first-serve principle, therefore the GPU Tracking algorithm is not completely deterministic. To test the GPU tracker data from Monte-Carlo simulations is used. Figures 6 and 7 show the efficiency as well as clone and fake track rations for CPU and GPU tracker respectively. It can be concluded that the GPU tracker is in no way inferior to the CPU version.

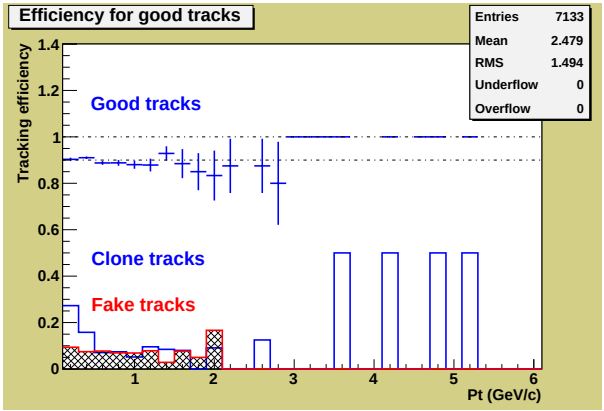

Fig. 6. Tracking efficiency of CPU Tracker

VI. Performance

While porting the tracker, the memory model was slightly changed. This resulted in more locality for optimal usage of available GPU memory bandwidth and also had positive cache effects on the CPU. Because of this and other optimizations, that were applied the new tracker code performs better by a factor of two on modern CPUs (benchmarked using 3.2

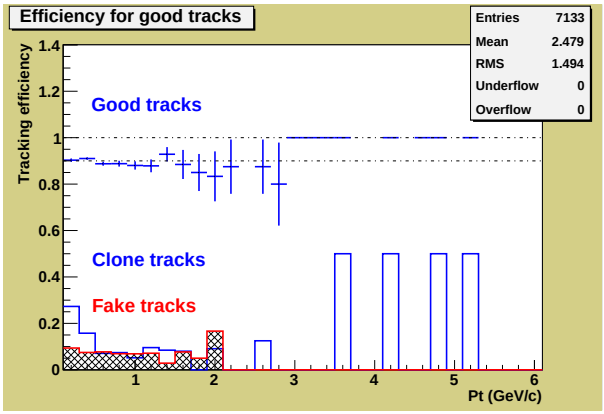

Fig. 7. Tracking efficiency of GPU Tracker

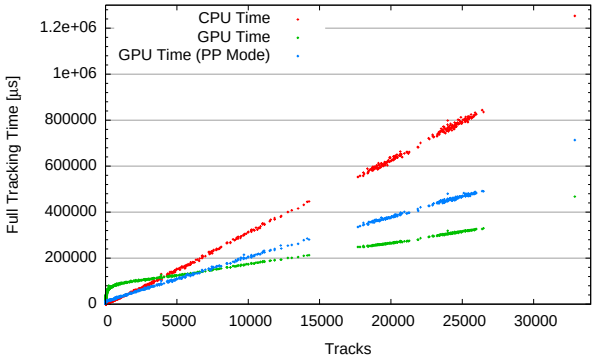

Fig. 9. CPU / GPU performance for different event sizes

GHz Intel Nehalem, 8 threads and data from monte-carlo-simulation) while the GPU version surpasses the processor by another factor of 3.3 for central lead lead collisions (Fig. 8).

Fig. 9 shows the performance of the CPU and GPU tracker on monte-carlo events of different size. Both trackers show a linear dependency on input size while the GPU tracker has a rather large offset. As a proof of concept analysis a different GPU tracker variant was created (pp-mode GPU tracker). However at both vacant event sizes for pp and heavy ion collisions either the CPU or the former GPU tracker performs better. Obviously the performance benefit of the GPU tracker is the larger the larger the event size.

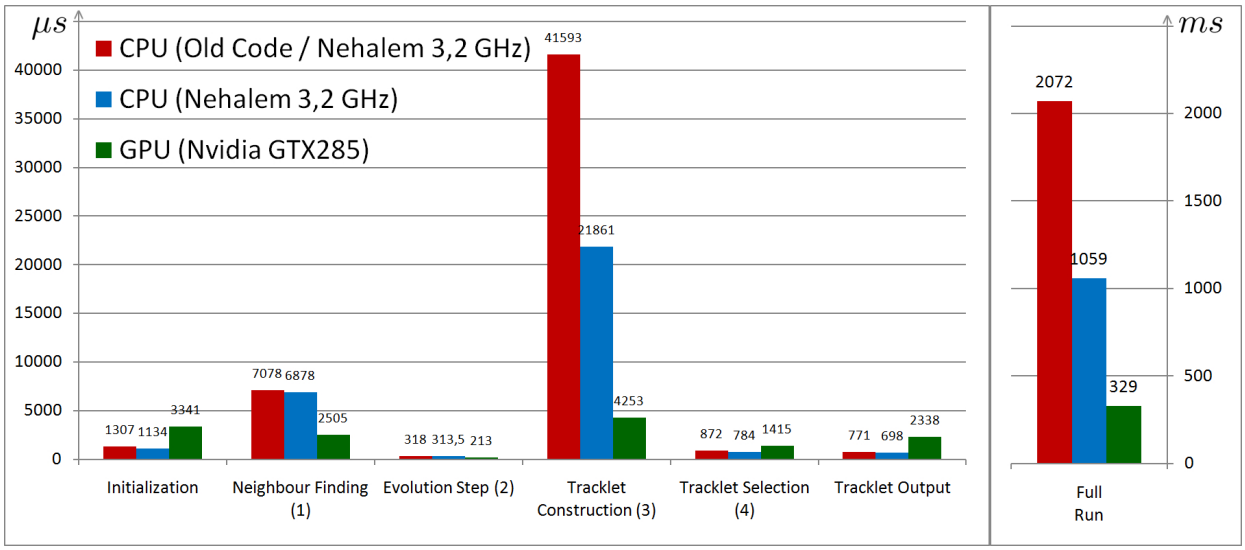

Fig. 8. GPU Tracker performance (Central pb-pb event)

VII. Summary

The ALICE TPC online tracking algorithm was successfully ported to NVIDIA CUDA. Several changes were made to the implementation of the algorithm. Finally, the GPU tracker easily outperforms the CPU implementation for heavy ion events while exactly maintaining its efficiency. Both versions share a common source code greatly improving the maintainability.

References

[1] ALICE collaboration. ALICE - Technical Proposal for A Large Ion Collider Experiment at the CERN LHC. *CERN/LHCC 1995-71*, 1995.

[2] K. Aamodt et al. The ALICE Collaboration. The ALICE Experiment at the CERN LHC. *JINST 3 S08002*, 2008. doi: 10.1088/1748-0221/3/08/S08002.

[3] S. Gorbunov et al. ALICE HLT High Speed Tracking and Vertexing. *Proc.17th Real Time Conference, Lisbon 2010*

[4] S. Gorbunov. On-line reconstruction algorithms for the CBM and ALICE experiments. Dissertation Thesis, Frankfurth Institute for Advanced Studies, in preparation

[5] S. Gorbunov, U. Kebschull, I. Kisel, V. Lindenstruth, and W.F.J. Mller. Fast SIMDized Kalman Filter based track Fit. Computer Physics Communications, 178:374 - 383, 2008

[6] R.E. Kalman. A new approach to linear Filtering and prediction problems. Trans. ASME-Journal of Basic Engineering, 82 (Series D) (1960) 35-45