Development and implementation of first level event selection process on heterogeneous systems for high energy heavy ion collision experiments

By

VIKAS SINGHAL

ENGG04201404002

Variable Energy Cyclotron Centre, Kolkata

A thesis submitted to the
Board of Studies in Engineering Sciences

In partial fulfillment of requirements
for the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

JULY, 2022

Homi Bhabha National Institute

Recommendations of the Viva Voce Committee

As members of the Viva Voce Committee, I certify that I have read the dissertation prepared by **Mr. Vikas Singhal** entitled "**Development and implementation of first level event selection process on heterogeneous systems for high energy heavy ion collision experiments**" and recommend that it may be accepted as fulfilling the thesis requirement for the award of Degree of Doctor of Philosophy.

Chairman – Dr. S. Pal	12/07/22
Guide / Convener – Dr. S. Chattopadhyay	12/07/22
Examiner – Dr. V. Bhatnagar	12/07/2022
Member – Dr. P. Y. Nabhiraj	12/07/22
Member – Dr. A. K. Dubey	12/07/22
External Member – Dr. S. Mukhopadhyay	12/07/22
Technology Adviser – Dr. V. Friese	18 July 2022

I hereby certify that I have read this thesis prepared under my direction and recommend that it may be accepted as fulfilling the thesis requirement.

Date: 12/07/2022

Place: VECC, KOLKATA

Signature
Guide

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

(Vikas Singhal)

DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree / diploma at this or any other Institution / University.

(Vikas Singhal)

List of Publications

Refereed Journal Publications

1. V. Singhal, S. Chattopadhyay, V. Friese. **"Investigation of heterogeneous computing platforms for real-time data analysis in the CBM experiment"**, **Computer Physics Communications** Volume 253, August 2020, 107190.
   DOI:10.1016/j.cpc.2020.107190.

2. V. Singhal, S. Chatterjee, V. Friese, S. Chattopadhyay. **"Development and implementation of a time-based signal generation scheme for the Muon Chamber simulation of the CBM experiment at FAIR"**, **Journal of Instrumentation** Volume 16, August 2021, P08043.
   DOI:10.1088/1748-0221/16/08/p08043.

Conference Proceedings

1. A. Jain, **V. Singhal**. **"Investigations of High Throughput Computing for Event Selection Process of MuCh@CBM"**, International Conference on Recent Advances in Interdisciplinary Trends in Engineering & Applications (RAITEA-2019), ssrn:3366339

2. P. Sen and **V. Singhal**. **"Event selection for MUCH of CBM experiment using GPU computing"**, 2015 Annual IEEE India Conference (INDICON), 2015, pp. 1-5,
   DOI: 10.1109/INDICON.2015.7443569.

3. **V. Singhal**, S. Chattopadhyay. **"High Energy Physics computing on heterogeneous platforms via OpenCL"**, DAE Symp. on Nucl. Phys (Vol. 58, p. 948), (2013).
   http://www.sympnp.org/proceedings/58/G53.pdf

4. **V. Singhal**, P. P. Bhaduri, S. Chattopadhyay, S. K. Aggarwal, **"Real time data analysis using GPU for High energy physics experiments"**, DAE Symp. on Nucl. Phys (Vol. 57, p. 972), (2012).
http://www.sympnp.org/proceedings/57/G57.pdf

Reviewed Progress Report

1. **"Investigation of pile-up effects in the CBM-MUCH using time-based simulations"**, GSI Scientific Report 2020 and CBM Progress Report 2020 (p. 95).
DOI:10.15120/GSI-2021-00421

2. **"Parallelisation of cluster and hit Finding for the CBM-MUCH"**, GSI Scientific Report 2017 and CBM Progress Report 2017 (p. 144).
DOI:10.15120/GSI-2018-00485

3. **"Time based MUCH Digitizer"**, GSI Scientific Report 2016 and CBM Progress Report 2016 (p. 167).
ISBN: 978-3-9815227-4-7

4. **"Event-building process from free-streaming data"**, GSI Scientific Report 2014 and CBM Progress Report 2014 (p. 114).
ISBN: 978-3-9815227-1-6

5. **"The CBM di-muon trigger on heterogeneous computing platforms "**, GSI Scientific Report 2013 and CBM Progress Report 2013 (p. 104).
https://repository.gsi.de/record/64893

6. **"First Level Event Selection for MUCH using GPU"**, GSI Scientific Report 2012 and CBM Progress Report 2012 (p. 97)
ISBN: 978-3-9815227-0-9

Other Publications

1. **"Commissioning and testing of pre-series triple GEM prototypes for CBM-MuCh in the mCBM experiment at SIS18 facility of GSI"**, A. Kumar, A. Agarwal, S. Chatterjee, S. Chattopadhyay, A. K. Dubey, C. Ghosh, E. Nandy, V. Negi, S.K. Prasad, J. Saini, **V. Singhal**, O. Singh, G. Sikder, J. de Cuveland, I. Deppner, D. Emschermann, V. Friese, J. Frühauf, M. Gumiński, N. Herrmann, D. Hutter, M. Kis, J. Lehnert, P.-A. Loizeau, C.J. Schmidt, C. Sturm, F. Uhlig and W. Zabołotny. **Journal of Instrumentation** Volume 16, September 2021, P09002. DOI:10.1088/1748-0221/16/09/P09002.

2. **"Event rate calculation after event building with mCBM data"**, S. Roy, **V. Singhal** and F. Uhlig, CBM Progress Report 2020 (p. 158). DOI:10.15120/GSI-2021-00421

3. **"An approach to estimate efficiency of a GEM chamber in the mCBM setup"**, E. Nandy, **V. Singhal**, S. Chattopadhyay, CBM Progress Report 2020 (p. 90). DOI:10.15120/GSI-2021-00421

4. **"Investigations on link loss in mCBM Data"**, S. Roy, **V. Singhal**, P. -A. Loizeau, XXIV DAE BRNS High Energy Physics Symposium 2020.

5. **"Testing of triple GEM prototypes for the CBM Muon Chamber system in the mCBM experiment at the SIS18 facility of GSI"** A. Kumar, C. Ghosh, S. Chatterjee, G. Sikder, A. K. Dubey, J. Saini, E. Nandy, **V. Singhal**, V. S. Negi, S. Chattopadhyay and S. K. Prasad. JINST **15**, C10020 (2020). DOI:10.1088/1748-0221/15/10/C10020

6. **"Update on mCBM data analysis of November/December 2019 beamtime"**, CBM Progress Report 2020 (p. 87).

DOI:10.15120/GSI-2021-00421

7. **"Response of the GEM chambers to varying intensity in mCBM data 2020"**, CBM Progress Report 2020 (p. 89).
   DOI:10.15120/GSI-2021-00421

8. **"Automatized noise separation technique for mMUCH data"**, S. Roy and **V. Singhal**, CBM Progress Report 2019 (p. 101).
   DOI:10.15120/GSI-2020-00904

9. **"Correlation between mMuCh hits and projected mTOF tracks in the miniCBM setup"**, CBM Progress Report 2019 (p. 105).
   DOI:10.15120/GSI-2020-00904

10. **"Implementation of Electronic FEB Id and channel Id for MUCH"**, CBM Progress Report 2019 (p. 75).
    DOI:10.15120/GSI-2020-00904

11. **"Response of mMUCH modules in the mCBM campaign 2019"**, CBM Progress Report 2019 (p. 71).
    DOI:10.15120/GSI-2020-00904

12. **"A Study of mMuCh Response at low and high intensity Pb+Au collisions"**, CBM Progress Report 2019 (p. 73).
    DOI:10.15120/GSI-2020-00904

13. **"Optimization of RPC detector segmentation and charge threshold in 3^{rd} and 4^{th} MUCH Station"**, CBM Progress Report 2019 (p. 76).
    DOI:10.15120/GSI-2020-00904

14. **"Testing large size triple GEM chambers with Pb+Pb collision at CERN SPS"**, Ajit Kumar, A. K. Dubey, J. Saini, **V. Singhal**, V. S. Negi, E. Nandy, S. K.

Prasad C. Ghosh and S. Chattopadhyay. XXII DAE High Energy Physics Symposium **1**, 13-16 (2018).
DOI:10.1007/978-3-319-73171-1

15. **"Implementation of RPC geometry and digitization in the 3^{rd} and 4^{th} MUCH station, CBM"**, CBM Progress Report 2018 (p. 163).
DOI:10.15120/GSI-2019-01018

16. **"Challenges in QCD matter physics –The scientific programme of the Compressed Baryonic Matter experiment at FAIR"** T. Ablyazimov, et.al., Eur. Phys. J. A **53**, 60 (2017)
DOI:10.1140/epja/i2017-12248-y

17. **"Testing of large size GEM detector with Pb+Pb collision at CERN-SPS"** A. Kumar, A. K. Dubey, J. Saini, **V. Singhal**, V. S. Negi, S. Mandal, S. K. Prasad, D. Nag, C. Ghosh and S. Chattopadhyay. DAE Symp. Nucl. Phys. **62**, 1006 (2017).
http://www.sympnp.org/proceedings/62/G8.pdf

18. **"Testing Pre-series prototype triple GEM chambers of CBM-MUCH with Pb+Pb collision at CERN SPS"**, CBM Progress Report 2017 (p. 75).
DOI:10.15120/GSI-2018-00485

19. **"Testing of MUCH-XYTER ASIC for the CBM-MUCH readout"**, CBM Progress Report 2017 (p. 77).
DOI:10.15120/GSI-2018-00485

20. **"Beam test of triple GEM prototypes with Pb+Pb collisions at CERN SPS"**, CBM Progress Report 2016 (p. 88).
ISBN: 978-3-9815227-4-7

Vikas Singhal 20/07/2022

(Vikas Singhal)

DEDICATED TO

My parents ***Mr. Sampat Raj, Mrs. Kaushalya Singhal,***

my wife ***Rashmi*** *and my kids* ***Agam, Aradhya****, and* ***Adhya***

ACKNOWLEDGMENT

My deep gratitude goes first to **(Prof.) Dr. Subhasis Chattopadhyay**, who expertly guided me through my doctoral research.

My sincere appreciation extends to my technical advisor **Dr. Volker Friese** for his mentoring, encouragement and early insights into the research work. I am obliged to my Doctoral Committee members for evaluation of my work and extending useful inputs during the reviews & at other times. I really appreciate the company of the VECC colleagues **Partha Pratim Bhaduri**, **Jogender Saini**, **Anand K. Dubey**, **Ekata Nandy** for nurturing my research acumen. I am extremely grateful to the entire CBM Collaboration.

I would like to thank my friends **Mr. A. Kumar, Mr. S. Chatterjee, Ms. S. Roy**, graduate students at CBM for their useful inputs & discussions.

I would like to thank **Ms. Priya Sen**, **Mr. Arijit Tewary, Ms. Piyali Ganguly, Mr. Kishanu Sarkar, Ms. Aarushi Jain**, the students who attended training at VECC during my thesis tenure and we worked together on different computing aspects.

I would also like to thank **Prasun Singh Roy**, **Abhishek Seal**, **Ashique Iqbal** for supporting in running the test setups.

A special thanks to my wife for all her sacrifices and prayers that made me sustained thus far. I would like to thank my kids who let me work. I would also like to thank all my friends who supported me in writing, and giving impetus to strive towards my goal.

Vikas Singhal 20/07/2022

(Vikas Singhal)

SYNOPSIS

The computing demands for modern experiments in high-energy particle and nuclear physics are becoming increasingly challenging. This holds in particular for experiments relying heavily on real-time data processing. For an example, real time processing requirement in the Compressed Baryonic Matter (CBM) experiment at the future Facility for Antiproton and Ion Research (FAIR), Germany is enormous. In High Energy collisions of protons or heavy ions many particles are generated with extremely low production rate. One such particle is J/ψ also known as charmonia and termed as J/Psi. J/Psi decays promptly into the dimuon channel of μ+ and μ- [1]. The multiplicity of the J/Psi is extremely small, therefore, to get reasonable statistics, CBM experiment intends to run with an unprecedented 10^7 collisions per second. A Muon Chamber (MuCh) system [2] will be employed for the detection of dimuon pairs originating from the nucleus-nucleus collisions. The MuCh system is being developed at VECC and it consists of alternating layers of segmented absorbers and detector stations. Each station consists of three detector layers. Due to very high interaction rates of up to 10 MHz, CBM will employ a free-streaming data acquisition with self-triggered readout electronics, without any hardware trigger. Efforts are concentrated towards the simulation of such raw data stream of messages termed as *"digi"* which will contain a global time stamp to generate free flow of data.

As collision rate is extremely high and J/Psi production is very low at FAIR energies (Lab Energy E_L = 10-40 AGeV), a real time selection process needs to be implemented to select only those events which are likely to contain J/Psi [3]. The major motivation of this work, towards the achievement of this goal could be itemized as below,

1. No algorithm is readily available for selection of events and based only on the hits, for particle interaction rate of the order of 10 MHz, on the detector. Therefore a detailed study needs to be carried out for development of a first level event selection algorithm. The algorithm also has to be optimized further to cater to the above unprecedented interaction rate.

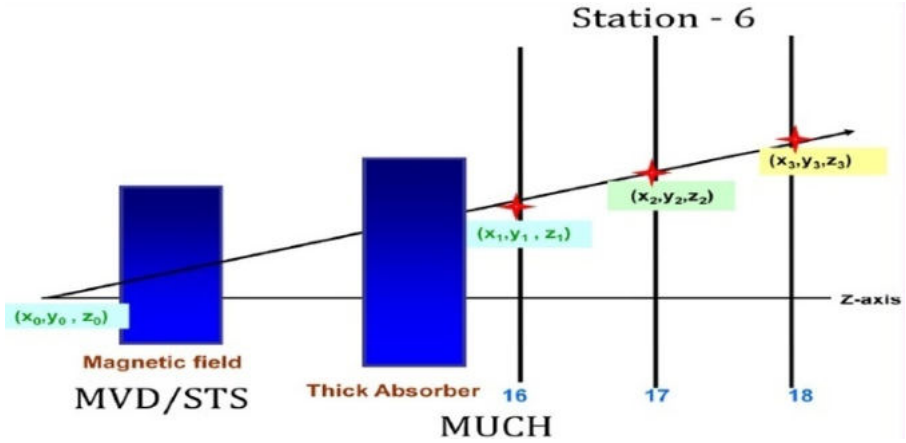

Figure 1: Pictorial representation of straight line fitting of 3 space points at the last station of MuCh.

2. To cope up with an event rate of 10^7 events per second, event selection algorithm; therefore, should deploy on the emerging technologies & platforms or their combinations to achieve its best possible performance.

3. A time based real data stream scenario is to be simulated. Studies for different interaction rates and the results are to be compared with the event by event data flow.

The work presented in this thesis titled "Development and implementation of First Level Event Selection process on heterogeneous systems for high energy heavy ion collision experiments" will demonstrate all the three requirements in detail and the thesis is therefore broadly divided into three parts. In the first part, algorithmic development of first-level event selection process will be discussed. The second part will concentrate on achieving real-time execution of the first level of event selection process using heterogeneous computing. The third part will focus on the development of a time based signal generation such that a realistic time based data stream can be generated which resembles the real experimental data stream.

First-level event selection process:

To select candidate events in real-time which contain J/Psi, an event selection criterion is required. The signature for J/Psi ($J/\psi \rightarrow \mu^+\mu^-$) candidate events is a rather simple one. The two daughter muons, having high momenta because of the large q value of the decay,

traverse all absorber layers and reach the trigger station, while hadrons, electrons, and low-momentum muons will be absorbed. Since J/Psi decays promptly ($c\tau = 7.1 \cdot 10^{-21}$s), the decay products practically originate from the primary (collision) vertex, i.e., from the target. Owing again to the high momenta of the muons, their trajectories can be approximated by straight lines even in the bending plane of the dipole magnetic field, which has a bending power of 1 Tm (as represented in figure 1). The trigger station consisting of three detector layers, termed as L1, L2, L3 respectively, provides three position measurements for a muon, allowing to check the back-pointing to the primary vertex. The signature of a J/Psi candidate event is thus the simultaneous registration of two particles in the trigger station which can be extrapolated backward to the target. On basis of the above signature, using brute force approach following event selection process has been developed,

1. Create a triplet of hits in the trigger station for an event, with one hit from each layer.
2. For the triplet:
   (a) Fit the hits of the triplet plus the event vertex (0, 0, 0) by a straight line as represented in figure 1. Perform fitting for both the $x - z$ (bending) plane and the $y - z$ (non-bending) plane, i.e., $x = m_0 z$ and $y = m_1 z$.
   (b) Compute the Mean Square Deviation (MSD) of the triplet fit.
3. Repeat the same steps [1..2] for all possible triplets of hits in the trigger station, with one hit from each layer.
4. Mark the probable event is desirable if their MSD_{xz} and MSD_{yz} are within threshold (discussed later) and two such triplets are registered.
5. Repeat the same steps for all the event.

For getting the final selection threshold for both (MSD_{xz}) and (MSD_{yz}), analysed both (MSD_{xz}) and (MSD_{yz}) distributions for 3 different data sets (signal events, background events, and background with one embedded signal event). MSD threshold values have been computed. Computational complexity of the above brute force algorithm is O(n^3) for an event. Here n

is the number of pads fired on the last layer of the trigger station in an event, which is of the order of maximum of 0.05% of the total pads on the last layer for the maximum FAIR energy setup. After analysing all the parallel computing techniques and performance optimizations discussed in heterogeneous computing section, execution time for the event selection process using the Brute-Force algorithm reaches to about 10^5 events per second whereas CBM is intended to run at an interaction rate of 10^7 events per second, therefore the algorithm is revised, and based on the selection of the region of interest, a novel algorithm has been developed and termed as **"Selective Algorithm"** which is different in terms of finding the probable triplets as compared to that in the Brute-Force algorithm. Working principle of the revised selective algorithm is as below:

1. In an event, select one space point from the L3 layer.
   (a) For each selected point, calculate two dimensional (2-D) angle with the vertex for both xz and yz plan assuming fitting in straight line. (A priory a region of interest has been computed based on study of different data sets.)
   (b) Search in a vector of points from L1 L2 layers.
   (c) Mark the triplet is desirable if point from the L2 and L1 fall in the region of interest.
2. If two such triplets are found in any event then the event is the probable candidate of Jpsi particle.
3. Repeat the same steps for all the event.

The Brute-Force algorithm is based on processing all the combinations of three points from the last three layers (one from each layer), however, in the selective approach single vector is created for L1 and L2 layer the complexity of selective algorithm is O(n^2). To optimized further, in selective algorithm, only those space points are processed which fall under the pre-analysed tolerance range and it is optimized for minimal memory consumption to maximize execution throughput on the co-processors. Both the algorithms have been implemented for

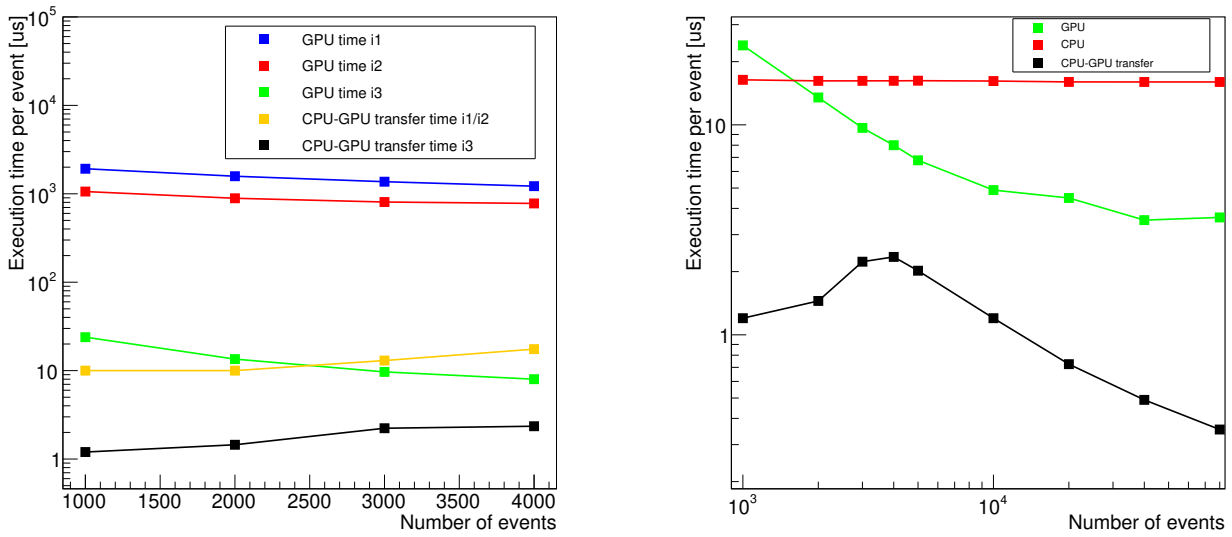

Figure 2: Processing time per event in microseconds as function of the number of events. The left panel compares the implementations i1, i2 and i3 with CUDA on the Tesla GPU. A comparison of the execution times on GPU (implementation i3) and on CPU (single-thread) is shown in the right panel.

the first-level event selection process. Comparison between both the algorithms has been performed. The achievement listed at the end of this synopsis shows that the developed novel selective algorithm is an optimized solution which can cope the CBM requirement within the available resources.

Heterogeneous computing:

For processing 10M events in a second within minimal resources, a detailed literature survey [4, 5, 6] has been performed to understand the state of the art methodologies. Modern computers come with a variety of concepts for concurrent data processing on many-core architectures, examples include, dedicated co-processors like NVIDIA or AMD GPUs, Many Integrated Core (MIC) by Intel such as Xeon Phi, the Accelerated Processing Unit (APU) of AMD, the Cell Processor of IBM and others. By using less powerful i.e. several cores of GPU, a heterogeneous CPU-GPU system can be used to accelerate the parallel processing. In a heterogeneous computing environment, different compute elements are interconnected to provide a variety of computational capabilities to execute tasks that have diverse requirements [6]. After survey of Flynn's Classical Taxonomy for heterogeneous architecture [7], it

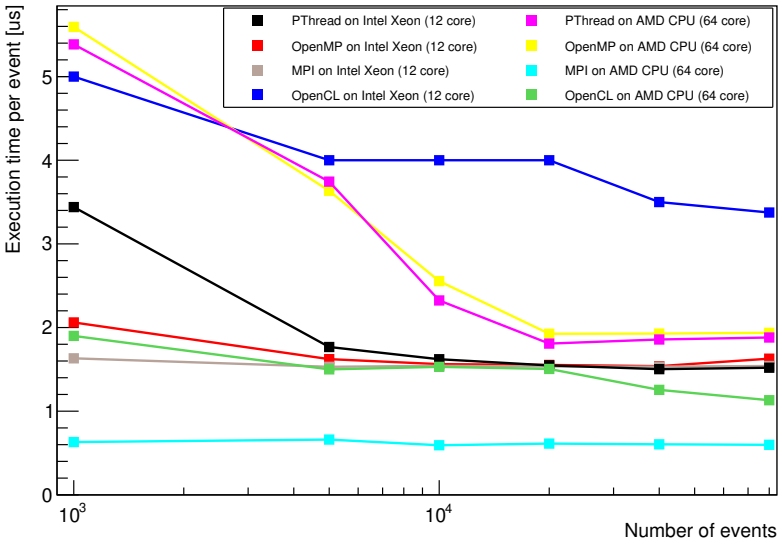

Figure 3: Execution time per event as function of the number of events processed at a time for the implementations with pthread, OpenMP, MPI and OpenCL on the Intel and AMD CPUs. The number of threads equals that of the available physical cores (12 for Intel, 64 for AMD).

is figured out that such systems are based on the SIMT (Single Instruction Multiple Thread) technology and come with a plenitude of processing units, allowing to run many threads in parallel. However, in order to make use of their computing potential, adequate programming paradigms are needed.

To achieve the fast execution of event selection process, heterogeneous computing based on many/multi core machine has been investigated in detail. The pure parallel programming paradigms Posix Threads (pthread), Open Multi-processing (OpenMP) and Massage Passing Interface (MPI) and heterogeneous parallel programming paradigms OpenCL (Open Compute Language) and CUDA [4] have been used for implementation & optimization of event selection trigger algorithms. TESLA and Quadro series of NVIDIA GPUs have been used as test-bench. The brute force event selection algorithm has been implemented using the CUDA API and then compiled with the NVIDIA compiler. Detailed analysis of the entire first level event selection process, with respect to execution time, memory footprint, CPU-GPU data transfer etc, has been performed. Memory has been arranged according to the GPU architecture and possible optimizations have been performed. Figure 2 (left), shows

the per-event GPU execution time for the various optimizations (i1, i2, i3) on a Tesla GPU and respectively the per-event CPU to GPU data transfer time and figure 2 (right) compares the single-threaded execution time on CPU. Multiple events are processed at the same time, one event being allocated to one thread.

Other than many core GPUs, detailed investigations have been performed using pthread, OpenMP, MPI and OpenCL for two different types of multi core hardware architecture Intel and AMD for FLES process. The results are shown in figure 3, showing the execution time for different configurations. Detailed investigation shows, the OpenCL is suitable option for heterogeneous computing environments.

Time Based Signal Generation:

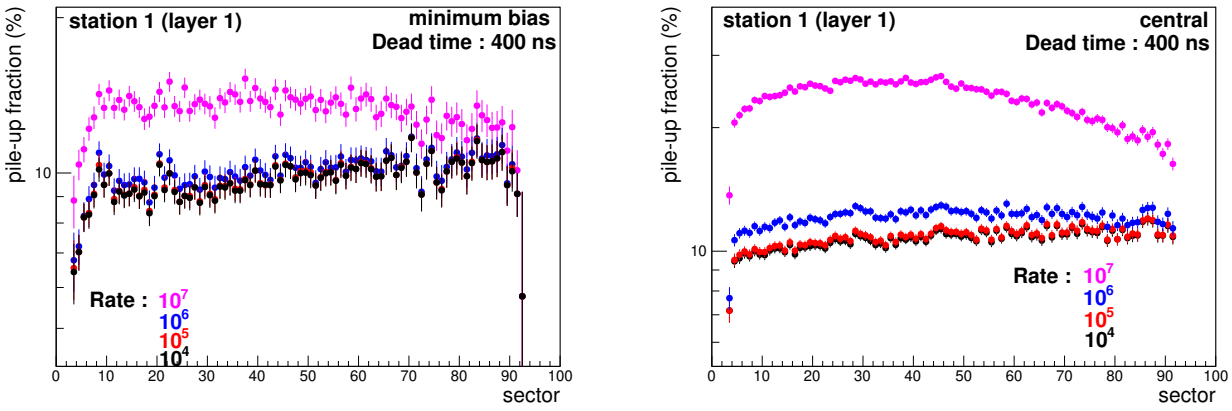

Figure 4: Pile-up fraction in each sector of the first layer of the first MuCh station for a dead time of 400 ns and for different interaction rates. The left panel shows the (realistic) case of minimum-bias events, the right panel the (artificial) case of central events only.

The data acquisition systems of most experiments in high-energy particle or nuclear physics are based on hardware triggers, where a signal generated by a suitable set of hardware indicates that a collision took place and triggers the timely readout of the front-end electronics. The hardware trigger thus defines an "event" as collections of moderate size data representing a separate single collision, which may then be either written to a permanent storage or subjected to further inspection and selection by higher-level triggers, e.g., on FPGA or in software. The software framework used for simulation and analysis of such experiments

are thus designed on an event-by-event scheme, where each event is treated as a separate and independent entity. But this triggered readout scheme is not feasible at CBM which intends to inspect up to 10^7 nuclear collisions per second, each producing several hundreds of particles to be registered in the detectors. These conditions will lead to use a trigger-less, free running data acquisition, where self-triggered front-end electronics register signals above a predefined threshold as caused by particles traversing the respective detectors and autonomously push the data forward. Such a system is not limited by latency, i.e. the time needed to generate a hardware trigger, but by data throughput bandwidth. It results in a continuous data stream in contrast to a series of events defined by the hardware trigger in conventional readout schemes. The association of raw data to physical events is based on precise time stamps. In this thesis, the software to analyse this data stream - both in real-time and offline will be discussed in detail. The time based simulation of the muon detector system MuCh incorporating response of detectors, front-end electronics and electronics noise etc., has been developed and performance of the same has been studied. Consequence of a self triggered free running data stream is inter-event pile-up due to dead-time of the underlying readout chip which is STS-MuCh-Xyter [8]. Figure 4 shows the pile-up fraction differentially for each sector in the first layer of the first MuCh station for different interaction rates, varying from 10^4/s to 10^7/s.

We enlist here major achievements in the thesis,

1. Development and implementation of first-level event selection (FLES) process for CBM-MuCh using brute force and a selective algorithms. Performed a comparison of performance between the two algorithms.

2. The developed event selection procedures suppresses the archival data rate by almost two orders of magnitude without reducing the signal efficiency, thus satisfying the CBM requirements for high-rate data taking [9].

3. A speed-up of 4.5 for event selection process using NVidia GPU is achieved in comparison with execution time of the optimized single-thread execution on CPU. (The

results are published in Computer Physics Communications journal.)

4. Results show that using the Brute Force Algorithm, about $3 \cdot 10^5$ events per second and using the Selective Algorithm about 10^7 events per second can be processed on a single GPU card of NVIDIA Tesla family. (The study has been presented in an international conference.)

5. Systematic study of heterogeneous architecture and different parallel computing paradigms have been presented and performance comparison of the algorithms on CPU, GPU using pthread, OpenMP, MPI, CUDA and OpenCL has been performed. It is found that the cross-platform OpenCL is a proper choice for heterogeneous computing environments typical for modern architectures, which combines CPU cores with GPU-like accelerator cards.

6. By the throughput obtained for the selection of J/Psi candidate event, it is determined that the computing demands of the CBM experiment can be achieved by properly making use of the parallel heterogeneous computing architectures.

7. Development of a time-based signal generation scheme for the Muon Chamber simulation by enabling interference of tracks from different events with small time separation, using an intelligent buffering procedure and a proper prescription for the treatment of signals arriving close in time in the same read-out channel (pad). (The results are published in Journal of Instrumentation.)

8. Integration of the MuCh simulation software with the cbmroot simulation framework to generate a free running data stream similar to that expected from the real experiment.

9. Development of noise generation framework for MuCh system such that the treatment of thermal, uncorrelated noise is included in the simulated data stream.

References

[1] Challenges in qcd matter physics –the scientific programme of the compressed baryonic matter experiment at fair. *The European Physical Journal A*, March, 2017. https://doi.org/10.1140/epja/i2017-12248-y.

[2] Subhasis Chattopadhyay et al., editors. *Technical Design Report for the CBM : Muon Chambers (MuCh)*. GSI, 2015. https://repository.gsi.de/record/161297.

[3] V. Friese. Computational challenges for the cbm experiment. *Mathematical Modeling and Computational Science. MMCP 2011. Lecture Notes in Computer Science, vol 7125. Springer, Berlin, Heidelberg 2012*. doi:10.1007/978-3-642-28212-6_2.

[4] Javier Diaz et al. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1369–1386, 2012. DOI:10.1109/TPDS.2011.308.

[5] Thematic CERN School of Computing. *Computing school organised by CERN with Collaboration with the University of Split, Croatia*, May 2015.

[6] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. L. Wang. Heterogeneous Computing: Challenges and Opportunities. *IEEE Computer*, 1993. DOI:0018-9162/93/0600-0018503.

[7] I. Ekmecic, I. Tartalja, and V. Milutinovic. A taxonomy of heterogeneous computing systems. *Computer*, 28(12):68–70, 1995. 10.1109/2.476202.

[8] K. Kasinski, A. Rodriguez-Rodriguez, J. Lehnert, W. Zubrzycka, R. Szczygiel, P. Otfinowski, R. Kleczek, and C.J. Schmidt. Characterization of the sts/much-xyter2, a 128-channel time and amplitude measurement ic for gas and silicon microstrip sensors. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 908:225–235, 2018. DOI:10.1016/j.nima.2018.08.076.

[9] V. Friese. The high-rate data challenge: computing for the CBM experiment. *Journal of Physics: Conference Series*, 898:112003, oct 2017. DOI:10.1088/1742-6596/898/11/112003.

Abstract of the Thesis

The computing demands for modern experiments in high-energy particle and nuclear physics are becoming increasingly challenging. This holds in particular for experiments relying heavily on real-time data processing. For an example, real time processing requirement in the Compressed Baryonic Matter (CBM) experiment at the future Facility for Antiproton and Ion Research (FAIR), Germany is enormous. In high energy collisions of protons or heavy ions many particles are generated and a few have extremely low production rate. One such particle is J/ψ known as charmonia. J/ψ decays promptly into the dimuon channel of μ+ and μ-. The multiplicity of the J/ψ is extremely small, therefore, to get reasonable statistics, CBM experiment intends to run with an unprecedented 10^7 collisions per second. A Muon Chamber (MuCh) system will be employed for the detection of dimuon pairs originating from the nucleus-nucleus collisions. The MuCh system is being developed at VECC and it consists of alternating layers of segmented absorbers and detector stations. Each station consists of three detector layers. Due to very high interaction rates, CBM will employ a free-streaming data acquisition with self-triggered readout electronics, without any hardware trigger.

The thesis presents the algorithmic development for selection of J/ψ events based only on the hits on the last station of the MuCh detector. A detailed study has been carried out for development of a first level event selection process. The algorithm also has been optimized further to cater to the unprecedented interaction rate. To achieve throughput of 10^7 events per second, event selection algorithm has been deployed on the emerging technologies & platforms or their combinations to achieve its best possible performance. Performance comparative study for FLES process has been presented using pure parallel and heterogeneous computing paradigms. After algorithmic development, a time based real data stream scenario is simulated for MuCh system with CBM collaboration. Studies of inter event and in-event pile up varying interaction rates have been done and the results, from simulated free running data stream, are compared with the event by event data flow.

CONTENTS

Contents **Page No.**

LIST OF FIGURES

No. **Title** **Page No.**

LIST OF TABLES

No. **Title** **Page No.**

Chapter 1

Introduction

In High Energy Physics (HEP) Experiments or during the high energy heavy ion collisions, different types of particles are produced with varying yields, some of them are produced very rarely depending on the collision energy & system. At relatively low energy collision like (Lab Energy E_{Lab} = 10 - 40 AGeV) one rarely produced particle is J/ψ meson, pronounced as J/Psi, which decays to dileptons with high branching ratios [1]. As the multiplicity of J/ψ is extremely small, therefore, to get a reasonable statistics, an experiment is required to deal with high collision rate [2]. In this direction, the Compressed Baryonic Matter (CBM) [3] experiment is an experimental setup at the upcoming Facility for Antiproton and Ion Research (FAIR) [4] which aims to collect data at the interaction rate of 10^7 collisions per second. As collision rate is extremely high and J/ψ production is very low at FAIR energies [5], a real-time selection process is needed to select only those events which are likely to contain J/ψ and the process should be executed at a speed equivalent to that of the interaction rate. For such an experiment, the software stack is required to simulate, reconstruct and analyse the data, in the realistic experiment scenario. For this, an approach is required which can not be based on the conventional 'event-by-event' simulation. In CBM, the data acquisition system will also provide a free flow data stream which is not separated collision by collision (i.e. event wise). To develop such a unique scheme of simulation, the

entire CBM collaboration, along with the other experiments at FAIR, has been working in this direction and each system is to provide realistic time based detector simulation incorporating response of detectors, front-end electronics and electronic noise.

The major motivation of this work, towards the achievement of this goal could be itemized as below,

1. No algorithm is readily available for selection of events and based only on the hits, for particle interaction rate of the order of 10 MHz, on the detector. Therefore a detailed study needs to be carried out for development of a first level event selection algorithm. The algorithm also has to be optimized further to cater to the above unprecedented interaction rate.

2. To cope up with an event rate of 10^7 events per second, event selection algorithm; therefore, should deploy on the emerging technologies & platforms or their combinations to achieve its best possible performance.

3. A time based real data stream scenario is to be simulated. Studies for different interaction rates and the results are to be compared with the event by event data flow.

The present thesis deals with all the three core aspects in detail. It will cover the development of a first level event selection process and then to achieve real-time execution of the first level of the event selection process, use of heterogeneous computing will be investigated and harness the available computing resources such that the requirement can be met [6]. This selection will be performed in real-time (almost on-line) and the main motivation is to archive the probable candidate of J/ψ events, thereby making the selection criteria not very stringent. Further selection of the events will be performed at a later stage at big computing farm during the detailed analysis of physics observables.

The aim is to get the events containing muons, decayed from the J/ψ at vertex. As muons are of high energy and high momentum, they will traverse almost in a straight line from the vertex and in the CBM muon setup they will reach the last layer of the Muon Chamber

System MuCh (described later). All other low momentum particles would be absorbed due to multiple absorbers as per construction of the Muon Chamber (MuCh) system [7] and most of the muons coming from J/ψ will reach the last layer. Background muons or the other particles which will reach the last layer may be secondaries or decay particles and will not satisfy the criterion of coming from the vertex. For selection of such events, two first level event selection algorithms, named Brute-Force and Selective, have been developed, analysed and implemented using heterogeneous computing platforms.

To achieve the fast execution of the event selection process, Heterogeneous computing based on many/multi core machines has been investigated in detail. The use of the Graphical Processing Unit (GPU), primarily used for graphics rendering, as a general purpose processing is becoming increasingly popular. This thesis describes how GPU computing can be used and beneficial in High Energy Physics (HEP) online computation. HEP computing deals with embarrassingly parallel problems as billions of events need to be processed using same algorithm (described in Sec. 3), therefore by using heterogeneous computing, online event selection can be achieved with higher event rate. The thesis discusses about the available computing architectures, available hardware brands, compilers, parallel paradigms and parallel libraries. In this thesis, the GPU, pure parallel paradigms, heterogeneous parallel paradigms among others will be discussed in detail with respect to the implementation and optimization of the event selection trigger algorithm.

The CBM experiment is designed to take data in nuclear collisions at very high interaction rates of up to 10 MHz and therefore will employ a free-streaming data acquisition with self-triggered readout electronics, without any hardware trigger. A simulation framework is being developed to cope with this requirement. For MuCh detector subsystem also a realistic digitization has been developed to provide a realistic simulation of the time-stamped data stream, which is a part of this thesis (in chapter 4 explained in detail). The implementation of the free-streaming detector simulation and the basic data related effects on the detector with respect to the interaction rate has been investigated in section 4.4.

1.1 CBM Experimental Setup

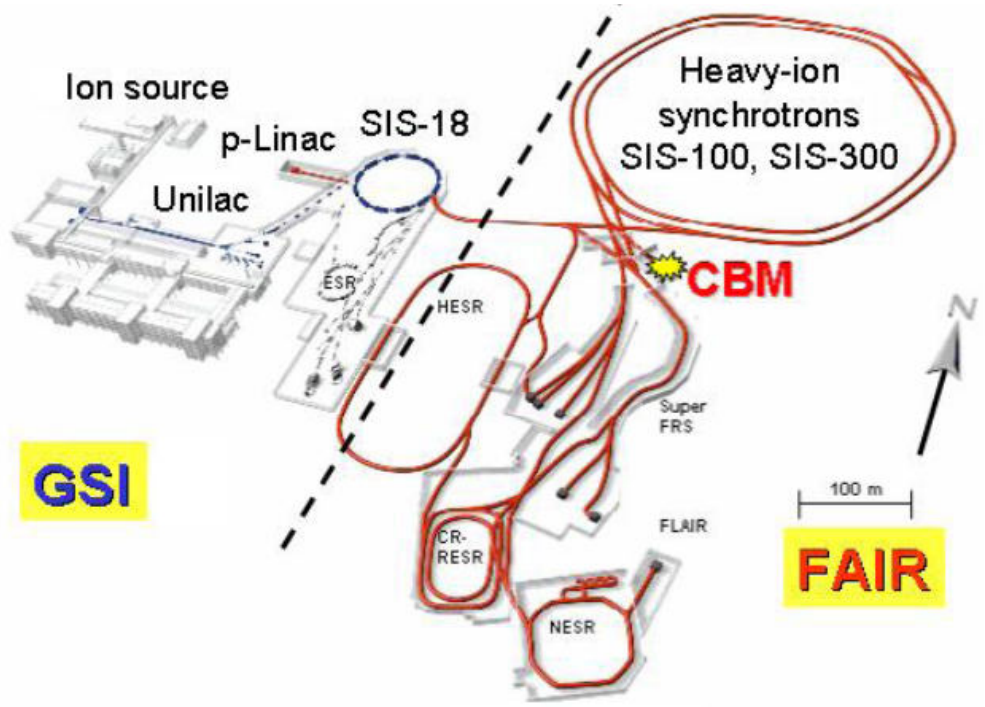

Figure 1.1: Schematic layout of the Facility For Antiproton and Ion Research (FAIR) complex at GSI with the CBM experiment using the beams from the SIS-100 and SIS-300 synchrotrons.

The Compressed Baryonic Matter CBM [3] experiment is a dedicated relativistic heavy ion collision experiment at the upcoming FAIR [4] facility at GSI Helmholtz Centre for Heavy Ion Research, Darmstadt, Germany. Figure 1.1 shows the layout of the FAIR facility. This figure shows two parts separated by dotted line. Left side of this figure shows the existing GSI complex in which existing accelerators like Unilac, proton linac and SIS-18 synchrotron are shown [8]. The right side shows the entire proposed FAIR facility which is presently under full-fledged construction and scheduled be ready by the year 2025. In the FAIR facility, SIS-100 and SIS-300 high energy synchrotrons are proposed to be built. In the first phase, SIS-100 will be commissioned and in future, SIS-300 will be commissioned as an upgrade plan. There are 4 experiments named CBM, NUSTAR (Nuclear Structure, Astrophysics and Reactions), APPA (Atomic, Plasma Physics and Applications), and PANDA (antiProton

ANihilation at DArmstadt). In Fig. 1.1, only the location of the CBM experiment is shown with yellow colour mark. In this experiment, highly energized ions (like Au) will be collided with a fixed target (Au plate) and after collision ($Au - Au$ collision), a large number of particles will be produced and they will pass through different detectors like Silicon Tracking Station (STS), Transition Radiation Detector (TRD), Time of Flight (TOF) detector. The CBM is a facility and different setups like hadron setup, electron setup and muon setup could be used. The Figure 1.2 shows CBM in the muon setup in which MuCh is in data taking position and Fig. 1.3 shows CBM in the electron setup with Ring Imaging Cherenkov (RICH) detector is in data taking position along with the other detector subsystems. Muon Chamber Detector System (MuCh) is shown in Fig. 1.4.

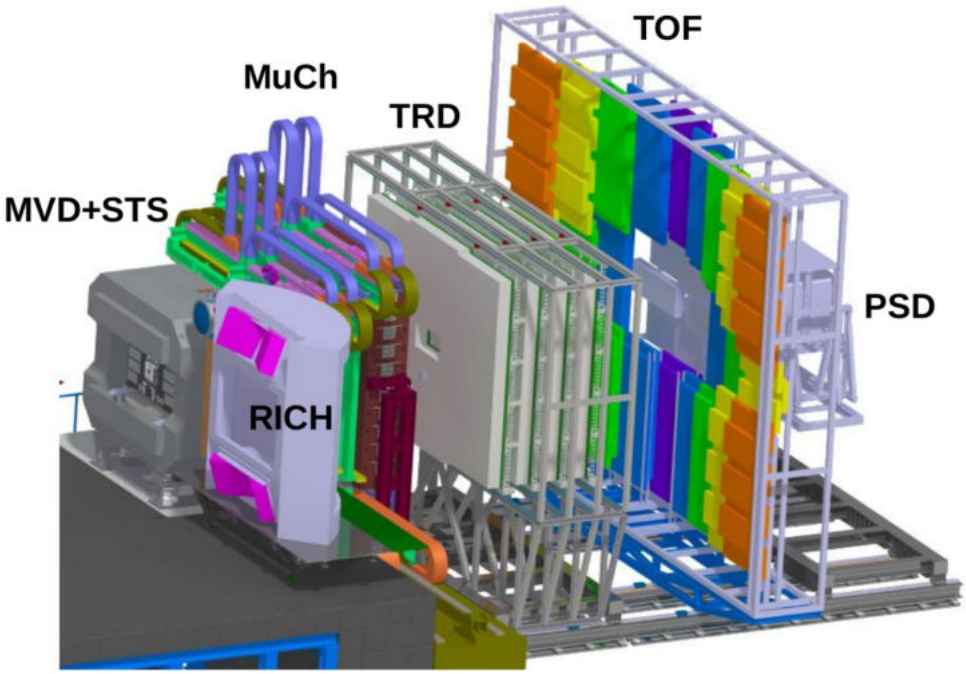

Figure 1.2: Schematic layout of The Compressed Baryonic Matter (CBM) muon setup with MuCh and RICH is in parking position.

1.2 MuCh and FLES

In this work, we discuss the use of MuCh for measurement of J/ψ, via their decay into the di-muon (μ^+ and μ^-) channel and as the production rate of J/ψ is extremely small at the CBM energy regime, therefore MuCh detector in the muon setup of CBM will run with extreme interaction rate of 10 MHz.

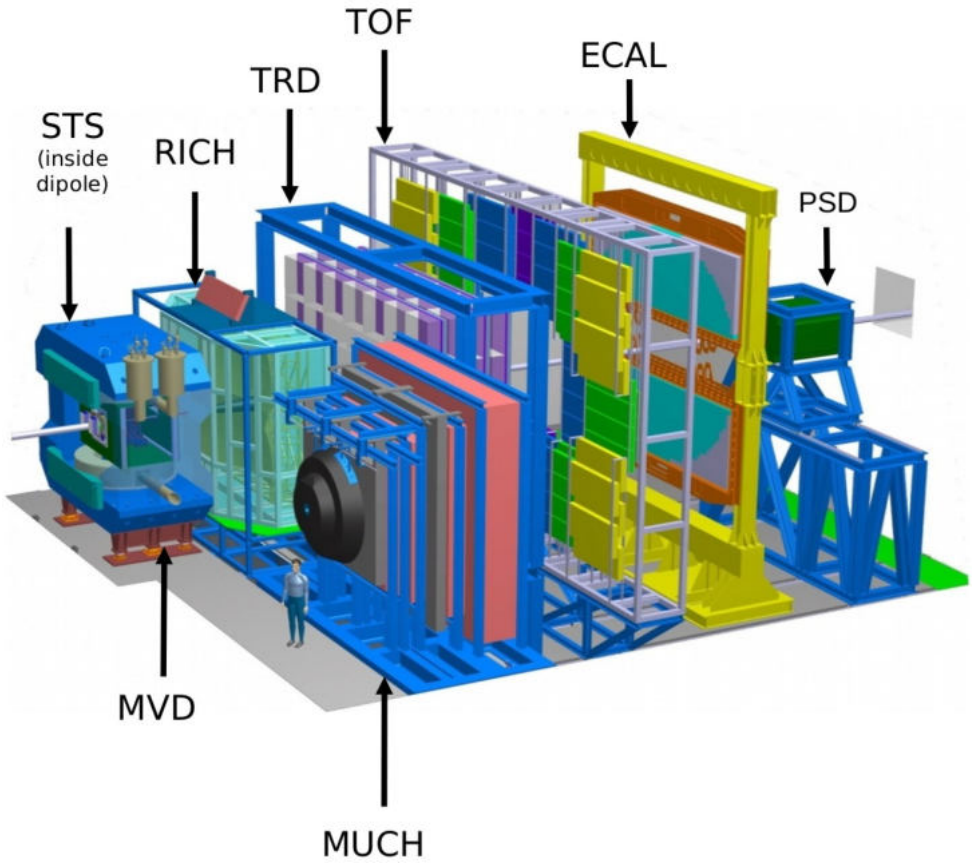

Figure 1.3: Schematic layout of The Compressed Baryonic Matter (CBM) electron setup with RICH and MuCh is in parking position.

Muon Chamber (MuCh) detector is being designed by a collaboration of Russian and Indian institutes. The muon detection system comprises of six iron slabs of different thickness, varying between 20 cm to 100 cm, with detector triplets inserted in the gap between two consecutive slabs. Each detector triplet between two consecutive absorbers is termed as one MuCh station. The total number of detector layers (also number of stations) will

vary based on the collision energy and the type of particles to be detected. The maximum energy for SIS-100 [8] will be 11A GeV and accordingly MuCh will comprise four detector stations with $3 \times 4 = 12$ layer configuration as shown in Fig. 1.4 and the maximum energy for SIS-300 will be 35A GeV, and corresponding configuration for MuCh will comprise six stations as shown in Fig. 1.5. The last three layers in the last station of the muon system is known as trigger station because our software trigger is based on the hits from these 3 detector layers only. This station is positioned after a one-meter thick iron absorber in the setup. As the multiplicity of J/ψ which decay into muons (μ^+ and μ^-) particle is very low and event rate is very high, therefore it is reasonable to store only those events which produce muons. Other events which do not resemble as probable candidate for J/ψ are known as background events. To reduce the background events, a first level event selection process is essential. After development of an event selection algorithm, another challenge is that it should be implemented such that 10^7 events can be processed within a second or less. In this thesis, we will describe in detail about a real-time event selection process and its implementation on heterogeneous hardware such that 10^7 events can be processed within a second. This event selection is known as online trigger software.

As per an estimate, the data acquisition system will provide more than 1 Tera Bytes per second of data stream, and the first level event selection process is a must in terms of reduction in the data volume by approx 3 order of magnitude from the storage point of view. For online reconstruction and the implementation of software trigger, a First Level Event Section (FLES) [9] computing farm is under development inside the Green IT Cube [10] at GSI, Darmstadt. FLES farm will comprise of data acquisition boards, input nodes and compute nodes. Raw data from data acquisition will directly land at the FLES. Due to first level of event building and first level of event selection will be performed on the FLES farm therefore it is termed as the First Level Event Selection farm. Discussions and implementations are ongoing for the tasks to be performed at the FLES level. All the event selection results presented in this thesis are based on the highest energy i.e. 35 AGeV and

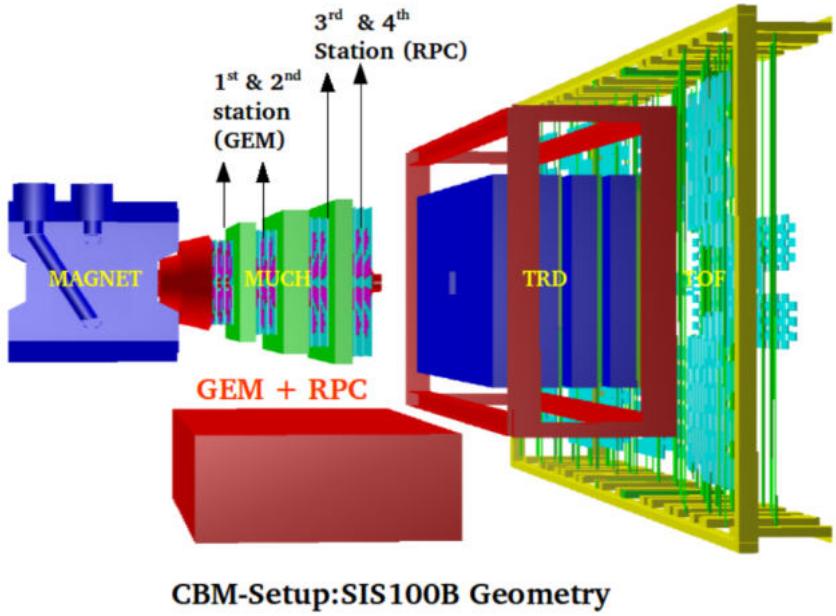

Figure 1.4: Schematic layout of the MuCh setup as per physics requirement of SIS-100 energies..

SIS-300 setup which will provide extreme conditions for computation.

1.3 Heterogeneous Computing

Computational power is a major requirement in high energy physics experiment which cannot be satisfied by conventional sequential processing on CPU. Parallel programming is one of the key options in this direction which allows many instructions to run simultaneously. Such a requirement can only be met using many and multi-core systems. To utilize such architecture, pure parallel computing paradigms like Posix thread, Open Multi-processing (OpenMP), Message Passing Interface (MPI) etc and heterogeneous parallel computing paradigms like Compute Unified Device Architecture (CUDA), and Open Computing Language (OpenCL) are available [11]. By using less powerful i.e. several cores of GPU, a heterogeneous CPU-GPU system can also be used to accelerate the parallel processing. GPU is a graphics processor or co-processor, mainly known as throughput processor which allows creating a large number of concurrently executing threads at very low system resource

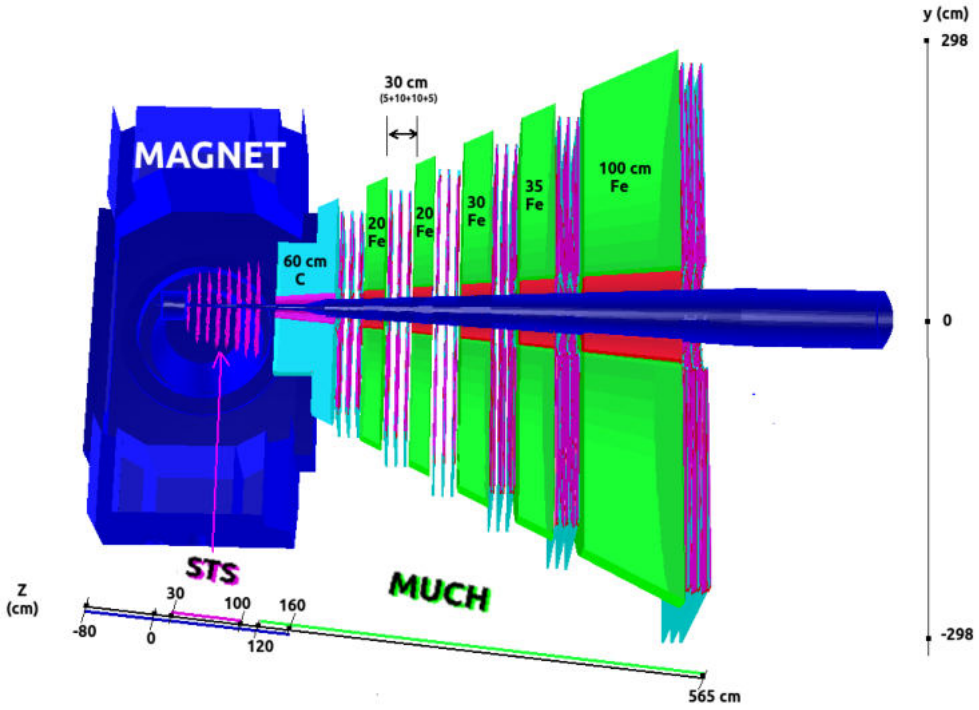

Figure 1.5: Schematic layout of the MuCh setup as per physics requirement of SIS-300 energies.

cost. Now a days it can also be used, other than graphics processing, for general purpose computing and therefore is termed as GPGPU (General Purpose Computing via GPU).

1.4 Time Based Simulation

The data acquisition systems of most experiments in high-energy particle or nuclear physics are based on a hardware trigger, where a signal generated by a suitable set of hardware indicates that a collision took place and triggers the timely readout of the front-end electronics. The hardware trigger thus defines an "event" as collections of moderate size data representing a separate single collision, which may then be either written to a permanent storage or subjected to further inspection and selection by higher-level triggers, e.g., on Field Programmable Gate Arrays (FPGAs) or in software. The software framework used for simulation and analysis of such experiments are thus designed on an event-by-event

scheme, where each event is treated as a separate and independent entity.

Several next-generation experiments, however, face the difficulty that such a triggered readout scheme is not feasible, e.g., due to high interaction rates and/or complex trigger topologies which are not well suited to be evaluated in hardware logic and are time consuming (CBM experiment is an example of such a scenario). CBM intends to inspect up to 10^7 nuclear collisions per second, each producing several hundreds of particles to be registered in the detectors. These conditions will use a trigger-less, free running data acquisition, where self-triggered front-end electronics elements register signals above threshold caused by particles traversing the respective detectors and pushes the data forward [12]. Such a system is not limited by latency, i.e. the time needed to generate a hardware trigger, but by data throughput bandwidth. It results in a continuous data stream in contrast to a series of events defined by the hardware trigger in conventional readout schemes. The association of raw data to physical events is based on a precise time stamp associated to each measurement by a central timing system.

The software to analyse this data stream - both in real-time and offline - must cope with the experimental situation [13]. The same holds true also for the software to simulate the experiment, which must generate such a data stream from appropriate physics models of high-energy nuclear collisions. The simulation package thus has to convert a series of events into a stream of data incorporating response of detectors, front-end electronics and data acquisition to these events. The CBM software framework `CbmRoot` [14], based on the `FairRoot` simulation and analysis framework [15], provides the appropriate structures for this task, in particular the software emulation of the data acquisition collecting the free-streaming data from the detector front-ends [12]. These framework structures must of course be filled by the implementation of the actual detector response of the various CBM sub-systems. Therefore, for the time based simulation of the CBM muon detector system (MuCh) simulation software has been developed using signal buffering scheme.

After this detailed introduction, chapter 2 will present the event selection signature,

algorithms and validation of algorithms. Chapter 3 will demonstrate in detail the implementation and optimization of the first level event selection algorithms on the many-multi core heterogeneous systems. Chapter 4 will describe the software implementation of the detector response for the MuCh detector. Verification and performance of the simulation is discussed in detail in section 4.4. Each chapter gives a summary and conclusions and also chapter 5 presents final conclusions and discusses about outlook on the development.

Chapter 2

First Level Event Selection

2.1 Introduction

High energy physics experiments fall into two categories, one is fixed target experiment in which particle beams are extracted from the accelerator and then collide with a fixed target like CBM and the other one is the collider experiments in which two particle beams coming from opposite directions collide with each other like in STAR, ALICE, CMS, LHCb, etc. In both types, one single collision is considered as one event. The events generated from these collisions are used to study different physics observables. On the basis of physics observable, the particular type of events need to be selected for which event selection process is required.

In the fixed target CBM experiment, the collision of heavy ions at beam energy range of 2 - 40 AGeV, a dense fireball is to be created. Multi-strange baryons, di-lepton pairs and charmed particles will be the diagnostic probes for such a medium [5]. All these observables will be identified via their decay products. The charmonia, a proposed key observable, can be measured via its decay into the di-muon channel μ^+ μ^- [1]. The average multiplicity of charmonium in these collisions is extremely small. These decay muons being weakly interacting, do not interact with the medium and make them a diagnostic probe. Due to low multiplicity of charmonia, data collection at an extreme interaction rate is required to

have a good statistics in a reasonable time. The low rates of these rare particles decide the experiment to plan to run at the required high interaction rate which is upto 10^7 events per second. This high interaction rate demands the experiment to perform with the cutting edge electronic technologies and the fastest computing processes. As the event rate is extremely high and J/ψ production is very low, therefore it creates another challenge of selecting events which are likely to contain J/ψ. Extreme interaction rate will generate approximately 1 TeraByte (10^{12} Bytes) per second of raw data stream [12] which is not feasible to store continuously at that rate on the permanent storage device. Thus, to achieve the goal, an event selection process or a trigger algorithm is required.

This chapter describes in detail the event selection process, i.e. the specific algorithms. This process of event selection will be deployed at the first stage after data taking and therefore is termed as the First Level Event Selection (FLES). Due to high interaction rate, hardware trigger is not feasible therefore this FLES algorithm is also known as software trigger algorithm. All the detector systems of CBM experiment are working with self-triggered electronics only.

2.1.1 Trigger signature

The signature of events for J/$\psi \rightarrow \mu^+\mu^-$ is visualized in the Fig. 2.1. The two daughter muons, having high momentum because of the large q value of the decay, traverse all absorber layers and reach the trigger station, while hadrons, electrons, and low-momentum muons will be absorbed before hand. Since J/ψ decays promptly ($c\tau = 7.1 \cdot 10^{-21}$s), the decay products practically originate from the primary (collision) vertex, i.e., from the target [6]. Owing again to the high momentum of the muons, their trajectories can be approximated by straight lines even in the bending plane of the dipole magnetic field, which has a bending power of 1 Tm [16]. The trigger station consisting of three detector layers provides three position measurements, allowing to check the back-pointing to the primary vertex. The signature of a candidate event is thus the simultaneous registration of two particles in the

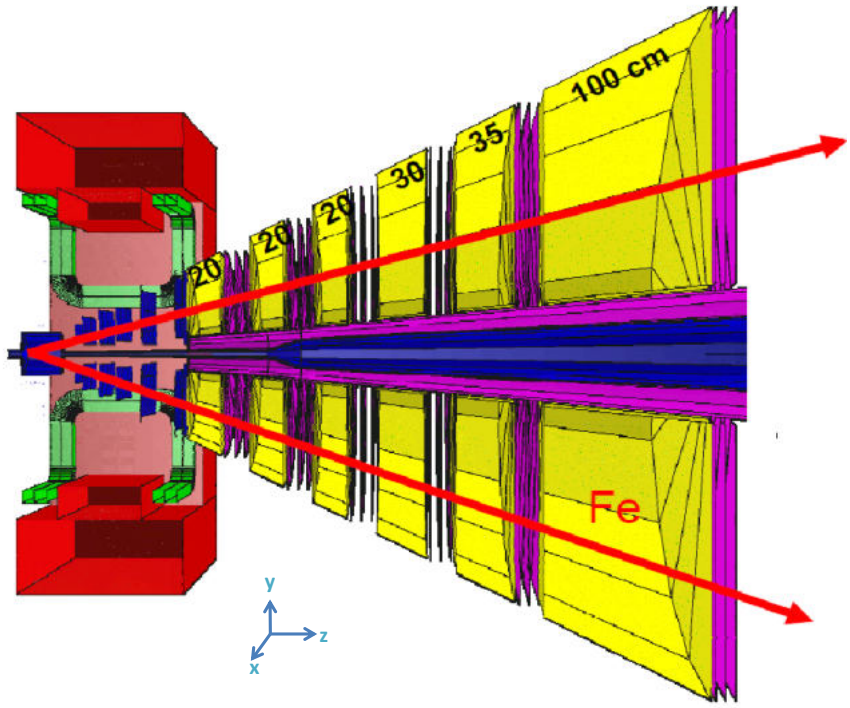

Figure 2.1: The CBM muon detection system with a pictorial view of a J/ψ decaying into μ^+ and μ^-. Triplets of tracking stations (magenta) are placed between absorber slabs (yellow).

trigger station which can be extrapolated backward to the target.

2.2 Software Environment

When we work in a large scale experiment like CBM, for doing simulation, reconstruction, material study and all the physics performance analysis, we need sophisticated modular software environment which consists of all the facility to do the required simulation, calculation, visualization and quality assurance. For this we used ROOT which is an open-source object-oriented data analysis framework. The ROOT framework utilises the latest C++ features and compatible with the C++14 and C++17 also. FAIR project software development team has bundled a toolkit named `FairSoft` comprising ROOT and many external packages like CLHEP, GNU Scientific Library, BOOST and many more. `FairSoft` provides a complete software environment to perform the simulation, reconstruction and data

analysis. To include core services specific to FAIR, desired physics analysis and experiment detector simulations, two more packages named `FairRoot` [15] and `CbmRoot` [14] have been developed and customised for the FAIR and the CBM experiment respectively. `FairRoot` provides common software environments for all the FAIR experiments and `CbmRoot` is a collection of libraries which enable all CBM related geometry, classes and parameters. The entire software stack is designed to optimise the accessibility for beginners and developers and is flexible (i.e. able to cope with future developments), to enhance synergy between the different physics experiments at/or outside the FAIR project. The entire software framework is working on the concept of continuous development and continuous integration and it supports many systems and compilers. Users are free to run `FairSoft, FairRoot` and `CbmRoot` on his/her desktop, laptop and/or server of any make and on any operating system variant. Efforts are going towards compiling `FairSoft, FairRoot` and `CbmRoot` software on any hardware/platform. For continuous compatibility, automatic testing across different hardware and platforms are going on and the accumulated test results from different platforms are sent to a web server, such that, results are displayed on dashboards by which developers can immediately know if there is any compatibility issue on any of the different systems.

`FairSoft`, `FairRoot` and `CbmRoot` have been installed for performing simulation related work presented in this thesis, in particular, for MuCh event selection and time based development (later described in chapter 4). Using this software stack, I have created simulated data and/or performed analysis with the desired geometry and setups.

As discussed in Sec. 1.1, CBM will operate with 2 interchanging configurations i.e. di-muon (shown in Fig. 1.4) and di-electron (shown in Fig. 1.3). The di-muon configuration of the MuCh system will also use different set of detector configurations. To cater all the configurations, `CbmRoot` provides multiple setups named as `sis100_muon_lmvm`, `sis100_muon_jpsi`, `sis100_electron` etc. Moreover, GEANT3 and GEANT4 [17, 18] transport engines are supported for transporting the particles through the geometry. GE-

ometry ANd Tracking (GEANT) is the name of a series of simulation software designed to describe the passage of particles through matter, using Monte Carlo methods. The framework enables us to construct detectors and/or analysis tasks in a simple way, it also delivers some general functionality like event display, track visualization etc. Moreover an interface for reading magnetic field maps is also implemented. For the work presented in this thesis `FairSoft, FairRoot, and CbmRoot` have been extensively used.

2.3 Event selection

As explained in Sec. 2.1.1, the aim is to get those events having muon pairs, which decayed from J/ψ at the vertex in three dimensional spaces (x, y and z). The mean momentum of muons from J/ψ decay at SIS-300 energies will be around 6 GeV/c [2]. As muons are of high energy and high momentum, they will traverse almost in a straight line from the vertex though high magnetic field in the STS region and they will reach the last layer of the MuCh because of low interaction cross section. Due to 100 cm thick iron absorber (situated before the last station of MuCh shown in Fig. 2.1 in the SIS-300 setup, all other low momentum particles would be absorbed and most of the muons coming from J/ψ will reach the last detector layer. Background muons or other particles which will reach the last layer may be secondary particles or decayed from other particles and not likely to satisfy the criteria of coming from the vertex in a straight line. For event selection, first, a simple algorithm named "**Brute-Force algorithm**" has been developed based on all the combinations between hits on the last 3 layers. After detailed study on this algorithm, another algorithm has been developed named "**Selective algorithm**". Both the algorithms are based on the following physics signatures:

1. Owing to very small lifetime J/ψ decay into μ+ and μ- at the vertex (origin),

2. Due to their large momentum these muons travel approximately in a straight line.

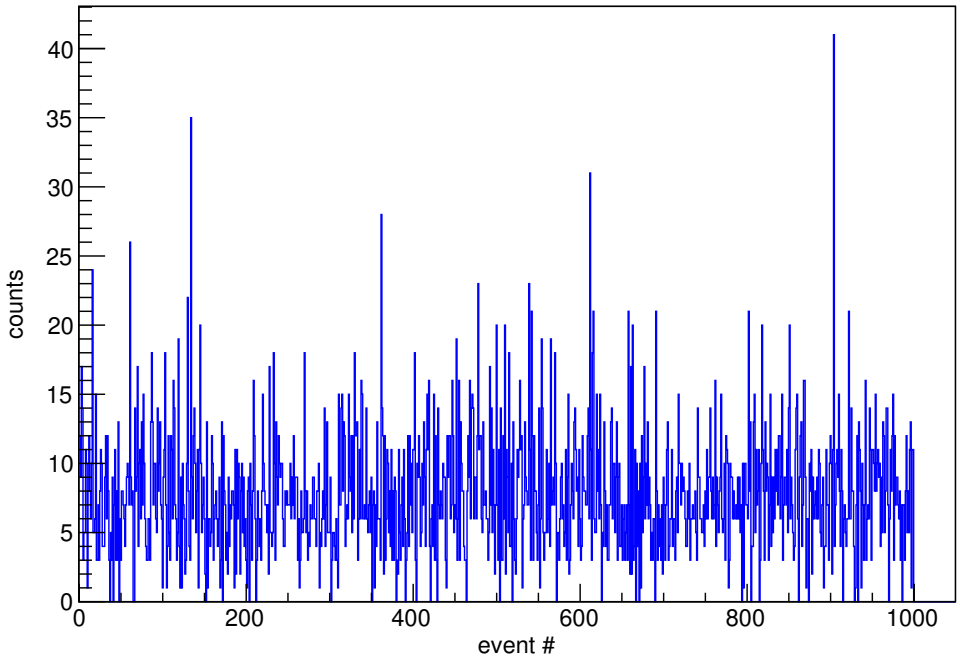

Figure 2.2: Event-wise hit distribution on the last MUCH station (combined of the last three layers);

The algorithms are therefore based on the straight line fitting of the hits in the last station. Fig. 2.4 depicts the straight line path of muons which will be produced by J/ψ and decayed at vertex.

To study the feasibility of these selection algorithms, a sample of background (minimum bias $Au + Au$) events (not containing any J/ψ decay) have been simulated in the CBM setup. The procedure of generating the simulated event in this work is discussed later. Fig. 2.2 shows the number of hits per event for the trigger station (combined for all three layers). The average number is about 45 for the trigger station and 15 for each of its layers . The dominant sources of this background are secondary muons originating from the weak decays of Λ and K^0_S between the target and the trigger station. Fig. 2.3 shows the spatial distribution of these background hits on the transverse plane $(x - y)$. The void area in the centre is not instrumented, since it hosts the vacuum pipe for the non-interacting beam [7, 19].

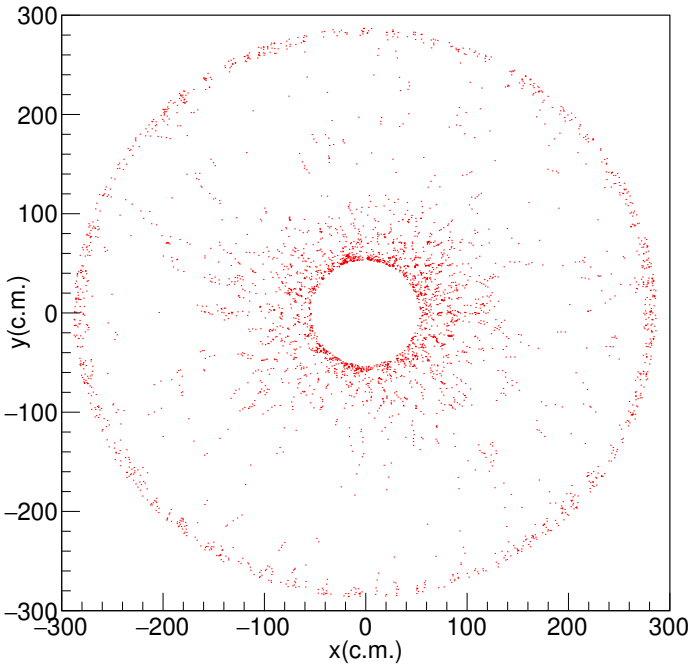

Figure 2.3: $x - y$ distribution of background hits in the trigger station for 1000 events. The accumulation at the periphery is due to secondaries not shielded by the absorber.

2.4 Brute-Force Algorithm

The Brute-Force algorithm is based on the approach of selecting triplets on the last station which fall on a straight line from the vertex out of all the possible triplets formed using 3 hits. The algorithm picks up three space points, termed as triplet from the last three layers of MuCh detector with one from each layer and makes all triplets of such combinations consisting of three points each. A threshold criterion (described later) has been designed to select the triplets. It may therefore be noted that a triplet means a combination of three space points from the last three MuCh layers (one from each layer) and fitted in a straight line passing through the vertex (0, 0, 0) as presented in the Fig. 2.4.

2.4.1 Implementation

External event generators are used in the `CbmRoot` framework to introduce the input event files for further transport simulation. In this work, we have used **U**ltra-**r**elativistic **Q**uantum **M**olecular **D**ynamics (UrQMD) [20] which is a microscopic model used for simulating (ultra) relativistic heavy ion collisions at the FAIR energy range and generate simulated events. This model is widely used to produce typical events for detector studies, as well as providing background events for physics studies. PLUTO [21] is a Monte Carlo simulation tool for hadronic physics. The vector meson decays were simulated with the PLUTO generator assuming a thermal source with a temperature of 130 MeV which is a tunable parameter. The background, consisting mainly of decays of charged pions and kaons, was calculated with the UrQMD event generator for the same system. Both the signal generated via PLUTO and the background generated via UrQMD are transported through the detector setup using `CbmRoot` employing GEANT3. One J/ψ signal particle is generated via PLUTO event generator and embedded during transport simulation in each event. For development of event selection, it is kept in mind that it needs to run online or on heterogeneous hardware, therefore the ROOT dependence is minimized. Now these generated simulated output space points corresponding to the interaction of particles through the medium were transported (moved) into a text file such that the input to further analysis is not dependent on ROOT file layout. The space points for 1000 $Au-Au$ collision events were read from a text file (in actual it will come from the online data stream) and depending on the value of Z, the space points for each event are stored into L1, L2 and L3 vectors (these are 3 different vectors for 3 different layers in the last station). Vector containers (from c++ stl) are efficient with respect to dynamic allocation of memory. The flowchart for the Brute-Force event selection process has been presented and discussed in ref [22].

In order to study the trigger algorithms and assess their performances, the following sets of simulated data were produced and studied:

D1. Signal events containing only one $J/\psi \rightarrow \mu^+\mu^-$ per event. The phase-space distribu-

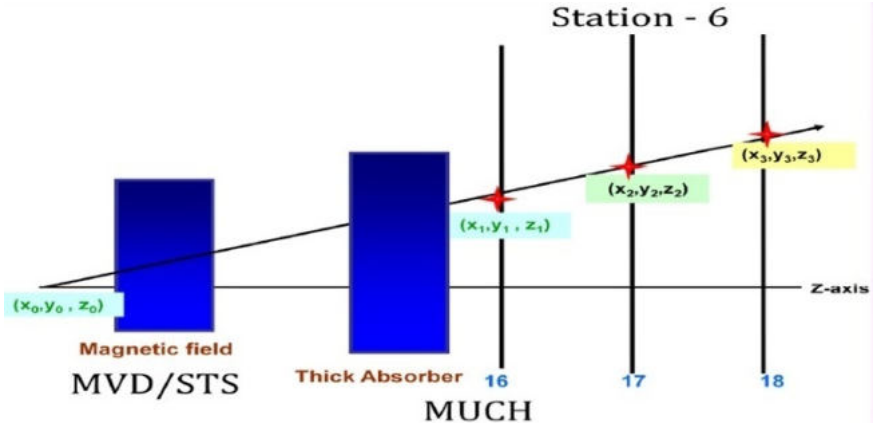

Figure 2.4: Pictorial representation of straight line fitting of 3 space points at last station of MuCh.

tion of the J/ψ were generated using the PLUTO generator [21].

D2. Background events (central $Au-Au$ at $p_{\text{beam}} = 35A$ GeV) generated using the UrQMD model [20].

D3. Background events with one embedded decay J/$\psi \rightarrow \mu^+\mu^-$ in every UrQMD event.

Following are the key steps for the Brute-Force event selection algorithm:-

1. Create a triplet of hits in the trigger station for an event, with one hit from each layer.
2. For the triplet:
   (a) Fit the hits of the triplet plus the event vertex (0, 0, 0) by a straight line as represented in Fig. 2.4. Performed straight line fitting in both the $x-z$ (bending) plane and the $y-z$ (non-bending) plane, i.e., $x = m_0 z$ and $y = m_1 z$.
   (b) Compute the Mean Square Deviation (MSD) of the triplet fit.
3. Repeat the same steps [1..2] for all possible triplets of hits in the trigger station, with one hit from each layer.
4. Repeat the same steps for 1000 Events distributed in 3 different data sets mentioned as D1, D2 and D3.

5. Analysed both (MSD_{xz}) and (MSD_{yz}) distributions for all combinations and a typical distribution of (MSD_{xz}) and (MSD_{yz}) are shown in Fig. 2.5.

6. For getting the final selection threshold comparisons have been made for different threshold values according to their MSD_{xz} and MSD_{yz}.

As per algorithm, detailed investigation has been performed for finding the optimal threshold criteria for the selection of probable triplets containing muon. A triplet is rejected if the MSD (per degree of freedom) is above the chosen threshold (0.03 for MSD_{xz}, 0.025 for MSD_{yz}). MSD distribution for the background is almost uniformly spread over the larger range as shown in Fig. 2.5. The MSD cut suppresses random combinations of hits as well as real triplets from secondary tracks. To illustrate, Fig. 2.5 shows the MSD distribution for the $x-z$ plane (upper) and the $y-z$ plane (below) for signal tracks (central $Au+Au$ events at $p_{\mathrm{beam}} = 35A$ GeV)(blue) with overlay of all triplets in background events (red) and a vertical (green) line indicates the cut value. The threshold is obtained from an optimisation procedure, resulted from the performance in terms of signal efficiency and background suppression. Computational complexity of the above Brute-Force algorithm is $\mathrm{O}(n^3)$ for an event. Here n is the number of pads fired on the last layer of the trigger station in an event as shown in Fig. 2.3.

2.4.2 Performance

A detailed study has been performed for selection of the mean square deviation (for xz and yz) of triplets with respect to following conditions:-

1. Those events which have at least one triplet.

2. Those events which have at least two triplets.

3. Those events which have at least one triplet which satisfy the threshold criteria.

4. Those events which have at least two triplets and satisfy the threshold criteria.

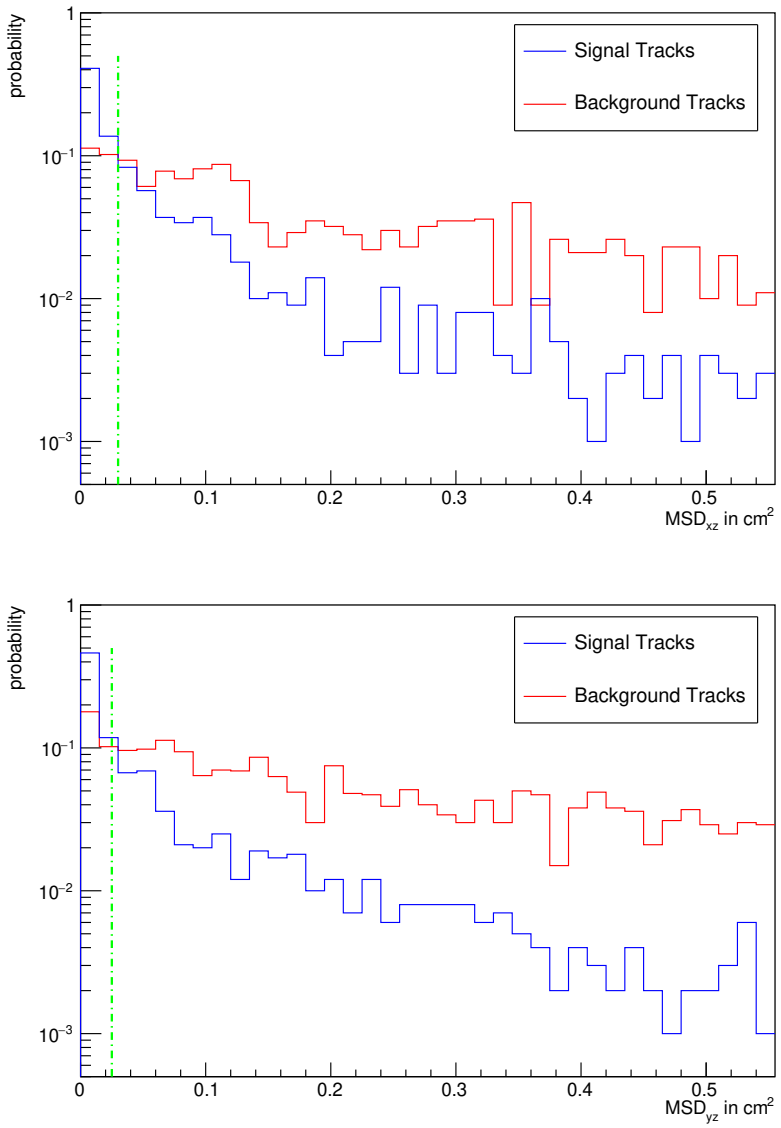

Figure 2.5: Upper: MSD_{xz} and lower: (MSD_{yz}) event normalised distribution for signal tracks in 1000 signal events (Au+Au at 35A GeV) overlay with all triplets in background events with vertical (green) cut line.

After collision, the probability of events with μ^+ and μ^- pairs is higher in the events with two and more triplets. So, efficiency/performance of the algorithm is calculated with those events with two or more triplets in the last station. In order to evaluate the performance of the trigger algorithm, the following performance figures are used:

a) efficiency (E): the fraction of embedded signal events selected by the algorithm;

b) efficiency under acceptance (EUA): the fraction of selected embedded signal events in which both decay muons have hits in all three layers of the trigger station;

c) background suppression factor (BSF): the ratio of all background events to the background events selected by the trigger algorithm.

The efficiency is mainly determined by the geometrical coverage of the muon detection system. Typical values are about 39 %. With the threshold values named above, the EUA is 85.4 %. The reason for it to be lower than 100 % is that one or both the signal tracks not passing the MSD cut. The BSF of 71.4 shows that the primary aim of the algorithm - suppression of a large fraction of the input data rate - is reached; the probability to find a chance pair of triplets passing the MSD cut is still not negligible.

Further background suppression will be achieved by full track reconstruction in the full CBM setup. This will provide the track momentum and more precise determination of the track impact parameter on the target plane, allowing to better separate primary from secondary muons. Full track reconstruction, however, is algorithmically involved [23] and thus requires significant computing resources. Owing to the first-level trigger algorithm described here, it needs only be applied on a data rate already reduced by a factor of about ~ 70.

Table 2.1: Computing performance tuning of a process

Level	Potential gains	Estimate
Algorithm	Major	$\sim$ 10x-100x
Source code	Medium	$\sim$ 1x-10x
Compiler Level	Medium-Low	10%-20%
Operating System	Low	$\sim$ 5%-20%
Hardware	Medium	10%-30%

2.5 Selective Algorithm

In the next chapter, we will discuss in detail about implementation of the Brute-Force algorithm on the heterogeneous platforms including in GPUs. In the Sec. 3.4, we will show that after analysing all the parallel computing techniques and performance optimizations, execution time for the event selection using the Brute-Force algorithm reaches to about 10^5 events per second whereas CBM is intended to run at an interaction rate of 10^7 events per second, therefore the event selection process needs to be improved in terms of computing speed. We analysed the Brute-Force algorithm and found that the processing of all possible combinations to form triplets is the most computing time intensive. To improve the computational complexity of the Brute-Force algorithm, we have investigated implementation of the triplet finding using matrix representation of fired hit coordinates, but the number of pads fired on the last layer of the trigger station in an event is of the order of maximum of 0.05% of the total pads on the last layer for the maximum FAIR energy setup. This resulted in huge memory requirement to store the hit coordinates for an event in the matrix form. As discussed in the chapter 3, GPU has limited memory, therefore the approach for storing hits in the matrix format is not optimal for the Brute-Force event selection process.

To increase the performance of any process with respect to the computation time, it can be guided by estimates provided in Table 2.1 [24]. According to the table, algorithmic improvement of a process provides major potential gain compared to the other hardware/-software tuning, therefore, the algorithm is revised, and based on the selection of the region of interest, we have developed a new algorithm termed as **"Selective Algorithm"** which

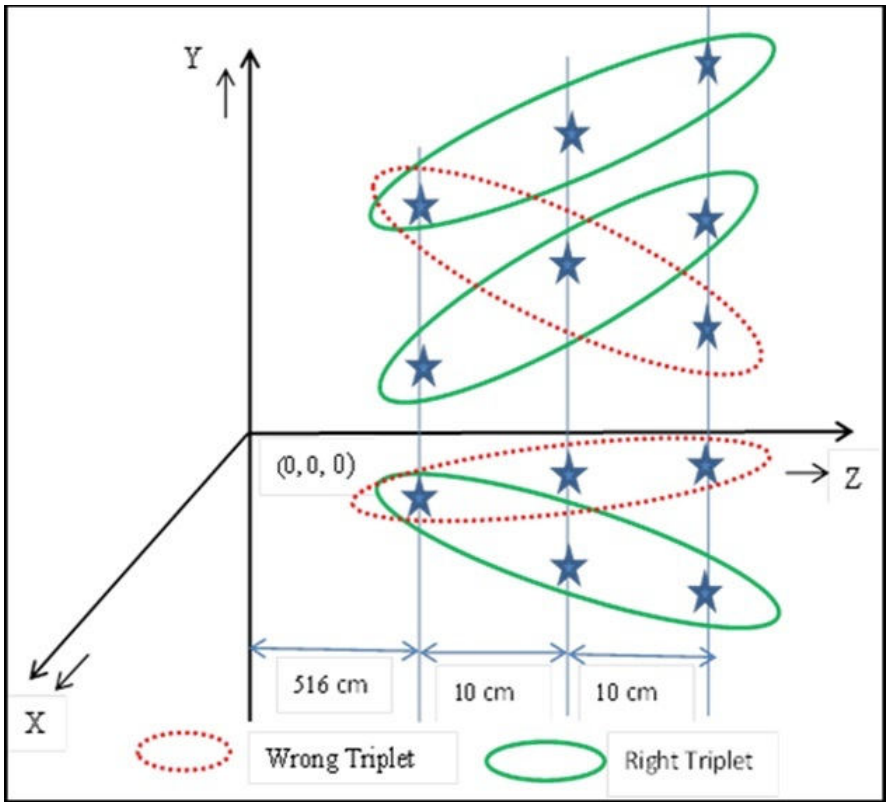

Figure 2.6: Pictorial representation of Selective Algorithm.

is different in terms of finding the probable triplets as compared to that in the Brute-Force algorithm.

Working principle of the revised selective algorithm is pictorially represented in Fig. 2.6. In the Brute-Force algorithm, all the combinations as shown in Fig. 2.6 using red and green oval shaped representations, have been processed for event selection. It is trivial that triplet combination represented by red oval shape will never fit in a straight line with the vertex $(0, 0, 0)$, therefore need not to be considered for further processing. Triplet represented as green oval mark might be fitted as straight line with the vertex. This logic is used in the Selective algorithm [25] and it is performed as below:

1. In an event, select one space point from the L3 layer.

   (a) For each selected point, calculate two dimensional (2-D) angle with the vertex for both xz and yz plan assuming fitting in straight line. (A priory a region of

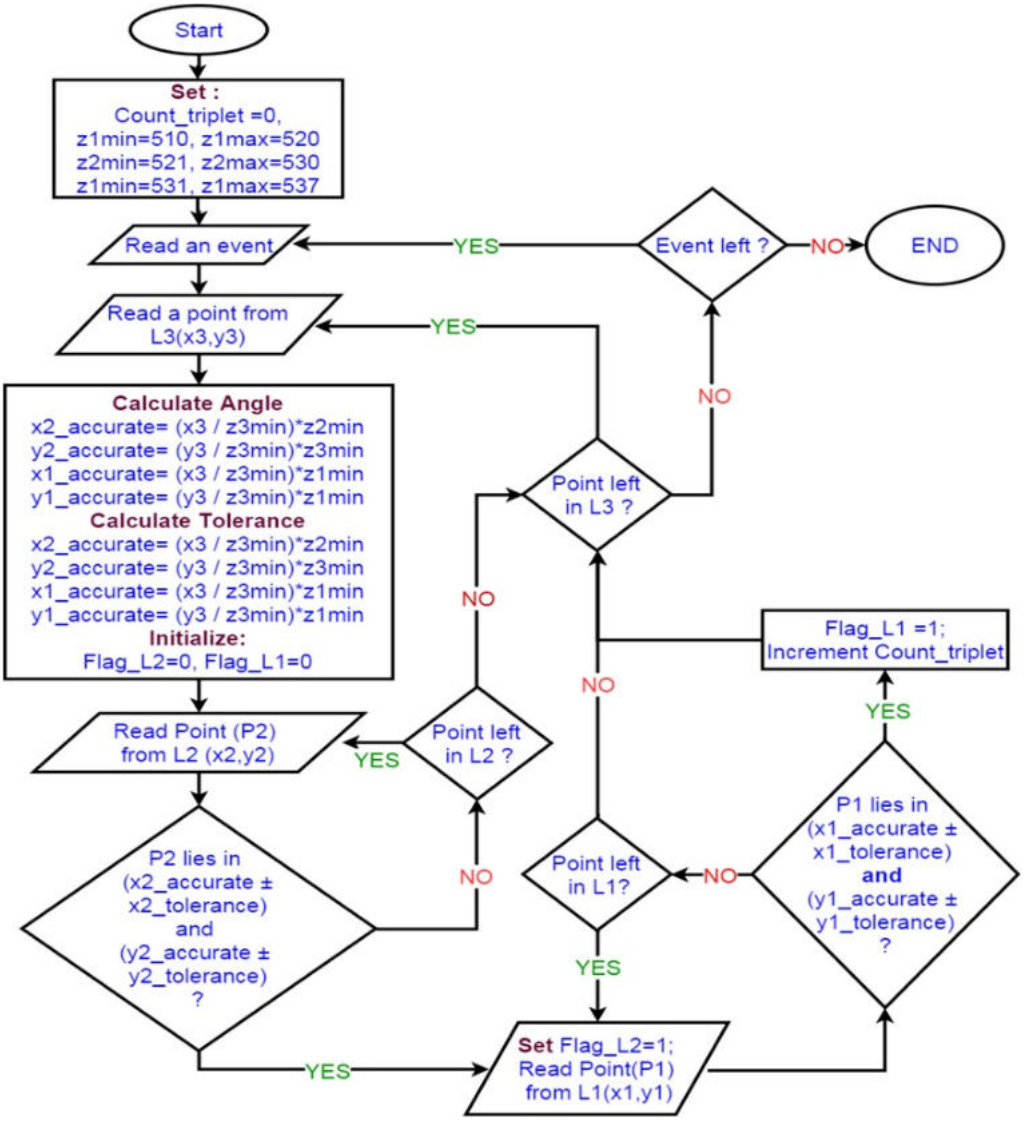

Figure 2.7: Flowchart for "Selective" event selection algorithm

interest has been computed based on study of different data sets.)

(b) Search in a vector of points from L1 & L2 layers.

(c) Mark the triplet is desirable if point from the L2 and L1 fall in the region of interest.

2. If two such triplets are found in any event then the event is the probable candidate of J/ψ particle.

3. Repeat the same steps for all the event.

Figure 2.7 shows the flowchart for the Selective algorithm and Fig. 2.8 shows the pictorial representation of triplet selection using the tolerance regions for (a) xz plane (b) yz plane separately.

As explained earlier, the Brute-Force algorithm was based on processing all the combinations of three points from the last three layers (one from each layer) and then performed straight line fitting for all the triplet combinations. However, in the selective approach single vector is created for L1 and L2 layer, therefore, the complexity of selective algorithm is O(n^2). To further optimise, in selective algorithm, only those space points are processed which fall under the pre-analysed tolerance range and it is optimised for minimal memory consumption to maximise execution throughput on the co-processors. Both the algorithms have been implemented for the first-level event selection process.

In Sec. 3.5, a comparison of per event execution time between both the algorithms is presented and it is demonstrated that using the selective algorithm, 10^7 to 10^8 events can be processed in a second using the experimental GPU used for this study.

2.6 Conclusion

In this chapter, requirement of event selection process and selection of probable event candidates which may contain J/ψ particle have been discussed in detail and the algorithms have

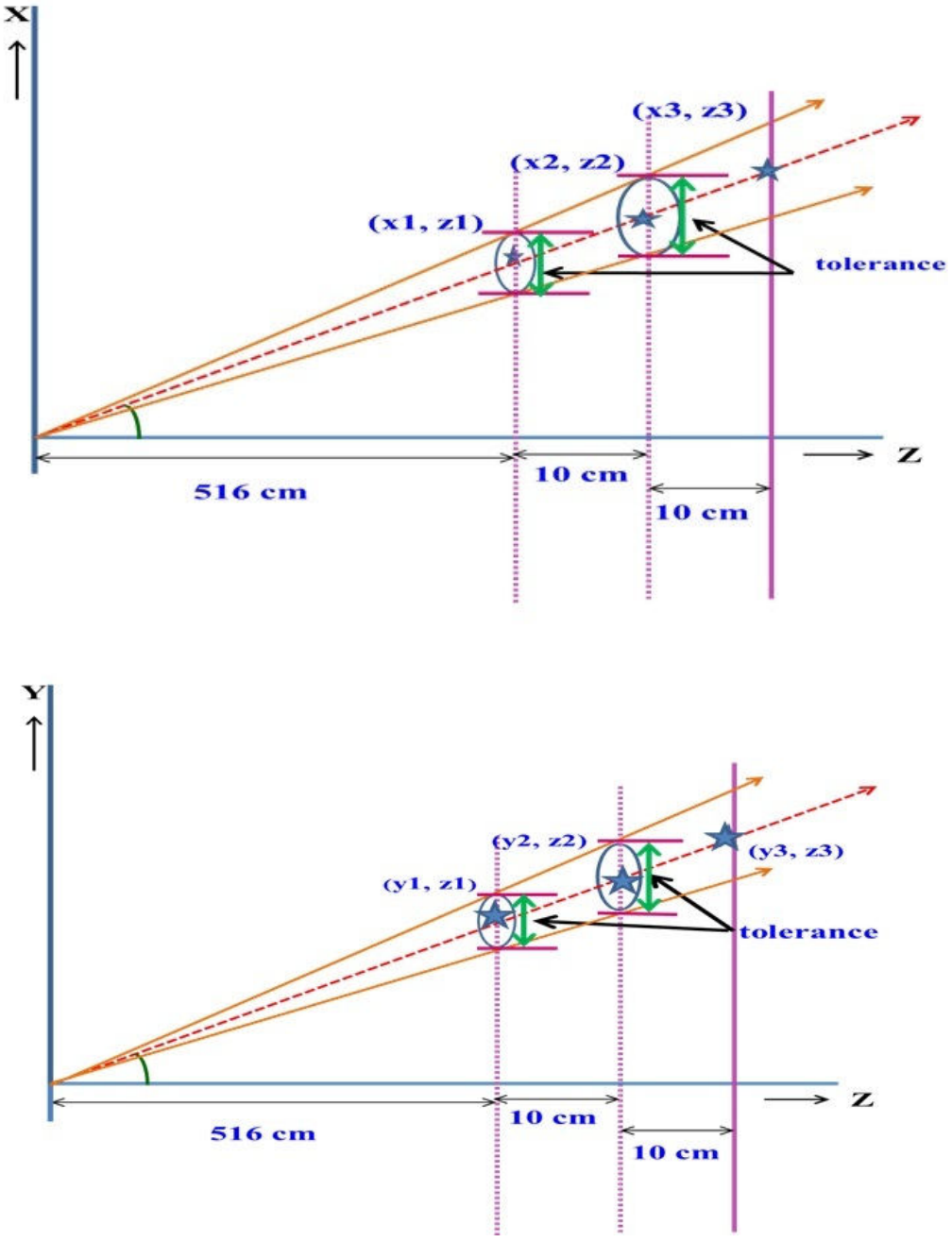

Figure 2.8: Pictorial representation of finding tolerance region for the Selective algorithm upper: xz plane, lower yz plane.

been discussed for the first level event selection process. Two different algorithms (a) the Brute-Force and (b) the Selective have been studied and flowcharts have been derived. Both the algorithms have been implemented on the software framework and tested with respect to the efficiency of selection, background suppression fraction and the computing time. It is seen that the requirement of high rate data taking for the CBM experiment can be met using the event selection algorithms. The algorithms suppresses the archival data rate by almost two orders of magnitude without reducing the signal efficiency. In the next chapter, we will discuss the implementation of both the algorithms on the many-multi core machines as the optimal computing performance with respect to execution time can be achieved using pure parallel computing paradigms and heterogeneous parallel computing paradigms.

Chapter 3

Heterogeneous and parallel computing paradigms

3.1 Introduction

HEP experiments produce a huge amount of data, therefore requiring non conventional computing approach for processing and further analysing. In this direction, multi-core architecture based processors or compute elements are being used for almost two decades. "Multi-core architecture" became essential as single core processor frequency hit the physical limitation of the clock oscillation frequency. After multi-core processors, on the similar line, many hardware developers have started building "many-core architecture" based dedicated processors other than CPU like NVIDIA GPU (Graphical Processing Unit), MIC (Many Integrated Core) by Intel named as Xeon PHI, AMD (earlier ATI) GPU. Manufactures have started experimenting on the CPU such that both many-core, multi-core architectures co-exist e.g. APU (Accelerated Processing Unit) by AMD, Cell processor by IBM among others. An APU contains both a CPU and a GPU on the same die allowing it to render and display images on screen. In this direction, fast pace developments are going on in the industry. All these various types of processors are based on different instruction-set

architecture (ISA). If a system hosts multiple processors based on different instruction-set architectures then it is known as heterogeneous computer and it is utilised for general computing then it is termed as **heterogeneous computing**.

In a heterogeneous computing environment, different compute elements are interconnected to provide a variety of computational capabilities to execute tasks that have diverse requirements [26]. There are many types of heterogeneous systems, including parallel, distributed, clusters, grids and clouds. These are found in industry, laboratory, government, academic, and military settings. An important research problem for heterogeneous computing is how to assign computation and communication resources to the tasks and to schedule the order of their execution to maximize some performance criteria, a process known as mapping or resource management. The factors that must be considered include machine and network loading, how well does the execution needs of a task match the computational capabilities of a machine, any inter-task communications, operating constraints, and the optimisation of the performance criteria.

When we talk about heterogeneous architecture, it is useful to list the Flynn's Classical Taxonomy for heterogeneous architecture [27]. Flynn's taxonomy distinguishes multiprocessor computer architectures according to the two independent dimensions, Instruction Stream and Data Stream. Each of these dimensions can have only one of two possible states: Single or Multiple.

1. SISD - Single Instruction Stream Single Data Stream
2. SIMD - Single Instruction Stream Multiple Data Stream
3. MISD - Multiple Instruction Stream Single Data Stream
4. MIMD - Multiple Instruction Stream Multiple Data Stream

Heterogeneity of computing came into picture prominently due to GPU, which targeted to maximise the computing performance or to accelerate the parallel processing. GPUs

are graphics processors, mainly known as throughput processors which allow creating a large number of concurrently executing threads at a very low system resource cost. As the name suggests, in the beginning, these processors were designed for graphics rendering purposes only. Subsequently it is observed that it can also be used for areas other than graphics processing e.g. for general purpose computing and started to be known as GPGPU (General Purpose Computing via GPU). All these hardware like GPUs, co-processors are based on SIMT (Single Instruction Multiple Thread) technology and come with hundreds of cores (specific lightweight cores) so that many such threads can run in parallel. On the contrary all the modern CPUs are based on MIMD technology and populated with a few general purpose cores. SIMT is very similar to Single-Instruction, Multiple-Data (SIMD). In SIMD, multiple data can be processed by a single instruction. In SIMT, multiple threads are processed by a single instruction in lock-step. Each thread executes the same instruction, but possibly on different data sets. It needs to be noted that, in terms of hardware architecture, a single core on CPU is different from single core on GPU though both are termed as core only.

After study of different system types and heterogeneous systems following is the overview of the presently available hardware as computing elements,

- CPUs (lightweight, heavyweight) – Mostly produced by Intel and AMD,
- GPUs (Graphics Processing Units) – Primarily produced by NVIDIA and AMD
- Co-processors – Mainly products of Intel eg. Xeon Phi
- APUs (CPU+GPU on the same chip) – Not suitable for high throughput computing
- DSPs (FPGAs) – Application specific hardware and has to be built according to the application.

In order to make use of the computing potential of the computing elements mentioned earlier, appropriate programming paradigms are needed. Parallel computing paradigms are

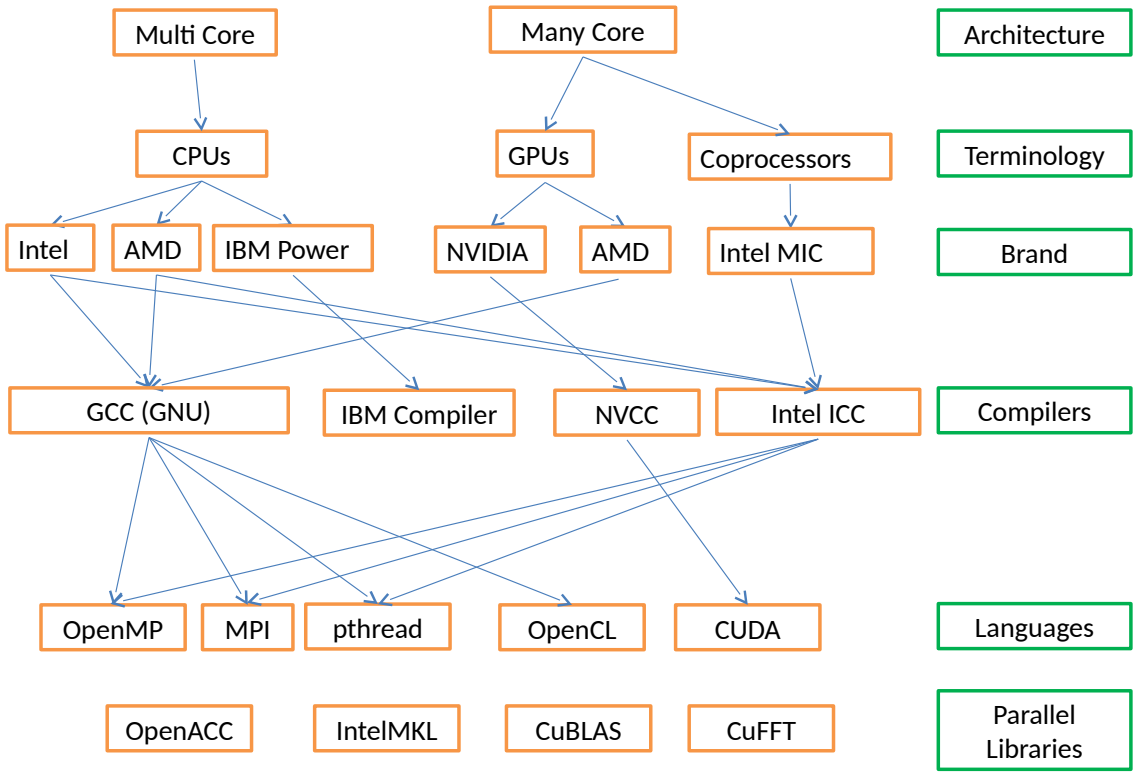

Figure 3.1: Overview of available hardware architectures and corresponding software stacks to utilize underlying systems.

broadly divided in two categories, (a) pure parallel programming paradigm for single type of compute elements and (b) heterogeneous parallel programming paradigm for multiple types of compute elements [11]. The pure parallel programming paradigms like Posix Threads (pthread) [28], OpenMP [29] and MPI (Message Passing Interface) [30, 31, 32] are available since long for utilizing multi-core CPU architectures like those of Intel, AMD, or IBM. For many-core architectures, Apple developed the OpenCL (Open Compute Language) managed by the Khronos group [33], and NVIDIA came with CUDA [34, 35]. CUDA is a proprietary parallel programming API that can be used for NVIDIA hardware only.

We enlist below the available parallel paradigms to utilize many-core and/or multi-core systems (list is not exhaustive):

- Posix Thread (p-thread): POSIX Threads, usually referred to as pthreads, is an execution model that exists independent of a language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time. The POSIX thread libraries are a standard-based thread API for C/C++. It allows one to spawn a new concurrent process flow.

- Open Multi-processing (OpenMP) : The OpenMP supports multi-platform shared-memory multiprocessing programming in C, C++, and Fortran, on many platforms, instruction-set architectures and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. It is an open source environment, based on shared memory, and useful for multi core computing.

- Message Passing Interface (MPI): Massage Passing Interface (Useful for clusters): Message Passing Interface (MPI) is a standardised and portable message-passing standard designed to function on parallel computing architectures. The MPI standard defines the syntax and semantics of library routines that are useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran.

- Compute Unified Device Architecture (CUDA): CUDA is a parallel computing platform and application programming interface (API) model created by NVIDIA. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing, an approach termed GPGPU. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. The CUDA platform is designed to work with programming languages such as C, C++, and Fortran.

- Open Computing Language (OpenCL): OpenCL, developed by Apple, is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors

(DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators.

- System Computing Language (SyCL): SyCL is a higher-level programming model to improve programming productivity on various hardware accelerators. It is a single-source domain-specific embedded language based on pure C++17. It is a standard developed by Khronos Group only and a successor of OpenCL.

- OpenACC: OpenACC is a programming standard for parallel computing developed by Cray, CAPS, Nvidia and PGI. The standard is designed to simplify parallel programming of heterogeneous CPU/GPU systems. It is an open programming standard for parallel computing and based on compiler directives.

In a nutshell Fig. 3.1 shows the available hardware software combinations. First two rows in Fig. 3.1 represent multi-many core architecture systems and terminologies used to denote them. Thereafter we present the available silicon developers who build such computing elements. After that, the compilers for compiling programs on these elements are shown. The second last row of Fig. 3.1 lists the parallel paradigms, which can be compiled using different compilers. In the last row, we have listed parallel libraries which may be used to parallelise the program. As none of the standard algorithms is used in our work, therefore, did not require any parallel library for the event selection algorithms.

In a parallel computing or computation, multiple processors work together to solve a given problem. The largest parallel machine has over a hundred-thousand processors with few millions of computing cores, and it is believed that machines with over ten thousand computing cores will be commonly available in the market very soon. Furthermore, with most ASIC manufacturers moving toward multi-core processors, most machines are parallel ones and therefore parallel computing is playing a pivotal role for exploiting the power of presently available infrastructure. While parallel machines provide enormous raw computational power, it is often not easy to make effective use of all this power. The problems

encountered in making effective use of a large number of machines are similar to those encountered in making a group of people work together. Challenges towards parallel computing can be understood with examples given below:

(i) People may spend much of their time talking to each other, instead of doing useful work. Communication between processors is quite expensive, compared with the CPU speed, so we need to pay attention to minimizing the amount of communication, otherwise much of the time will be spent on inter-processor communications, rather than on useful work.

(ii) In a group, a few people may only do much of the work, while others relax. Similarly, in a computation, the work load on different processors may differ. In order to make effective use of the parallel machines, we need to keep the workload balanced on all processors.

(iii) It may be difficult to decompose a problem so that people can work on different parts simultaneously.

For example, consider someone who wants to have dinner cooked, eat it and then have the dishes washed. It is not easy to speed up this process by hiring someone to cook, and another person to wash the dishes, because the three tasks are sequential; the food needs to be cooked before it is eaten, and the food needs to be eaten before the dishes are washed. Similar problems occur in parallel computations too, and the sequential parts of the computation can reduce the effectiveness of parallelisation substantially, however, HEP computations is trivial to parallelise at inter event level, rather intra event level. In a typical HEP computation, similar or almost the same computations need to be performed on hundreds of millions of events and computed information need to be collected for each event.

Main motivation behind parallelising any problem is to save time and maximise the utilization of all the underlying resources. During the past 20+ years, the trends indicated

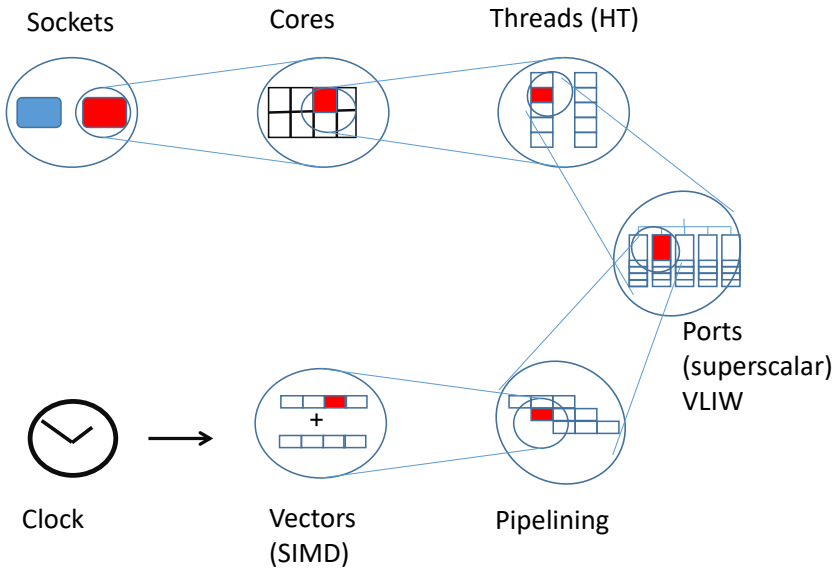

Figure 3.2: A modern computer architecture: which provides different dimensions to achieve high computing performance.

by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that parallelism is the future of computing.

When we talk about a general computing element like a single CPU, it comes into mind that what are the dimensions available in the CPU by which we can harness or exploit the computation. Fig. 3.2 shows seven dimensions of a modern computer architecture and by utilising these we can improve the computing performance. We have used these for optimising event selection algorithms e.g. optimisation using vectorisation, pipelining and compiler features have been performed and improvements obtained. However, in general, as discussed in Sec. 2.5 and presented in Table 2.1, to achieve one to two order of improvement in computation time for any problem, the underlying algorithms need to be optimised.

3.2 GPU for Event Selection

As described in introduction Sec. 1.2, GPUs are going to be used as basic computing building block along with CPUs for event selection work in the CBM Experiment and will be placed in the FLES. The First-level Event Selector (FLES) will be the central physics selection system in the CBM. Full event reconstruction will be performed online in FLES on the 1 TB/s input data stream [12]. In order to achieve the high throughput and computation efficiency, all available computing devices will be used, in particular FPGAs at the first stages of the system for aggregation and sorting of the subsystem hit messages, followed by heterogeneous systems comprising CPUs & GPUs, for the subsequent event reconstruction, selection and track reconstruction.

As per the motivation discussed in Sec. 1.3, after development of the event selection algorithm, major challenge is to perform the event selection on the heterogeneous systems such that 10^7 events per second can be processed. In this section, we explore the GPU execution process, architecture specific memory arrangements, the implementation and optimisation of the event selection process on NVIDIA GPUs. We compare different heterogeneous parallel programming paradigms (CUDA and OpenCL) on NVIDIA's Tesla and Quadro GPUs.

For the algorithms described in the previous chapter 2, we have tested implementations of both the algorithms on the following two heterogeneous platforms which are used as test bench for the entire study of this thesis:

S1. A Dell T7500 workstation comprising two Intel Xeon 2.8 GHz six-core processors with 2 GB/core RAM, together with two NVIDIA GPUs (Tesla C2075 [36] and Quadro 4000 [37]).

S2. An AMD-based HP Server with four AMD Opteron 2.6 GHz processors comprising 16 cores with 4 GB/core RAM (total 64 cores).

3.2.1 GPU execution process

GPUs are the co-processors and can not be used as a main processor on which operating system can reside. Machines based on heterogeneous architecture, GPUs or co-processors are termed as 'device' and CPUs are as a 'host'. Methods or functions which will be executed on the GPUs are termed as Kernel and are denoted using $\_\_kernel\_\_$ compiler directive.

To utilise a GPU, following basic steps need to be adapted as a work flow,

1. Program the code for CPU GPU including GPU kernel part.
2. Allocate GPU memory.
3. Transfer and copy needed data from host to device.
4. Execute program in parallel at GPU.
5. Transfer back the result or outcome from the device to the host.
6. Use the results for further processing.

3.2.2 GPU Architecture and TESLA C2075

In the hardware industry, NVIDIA is the major player for the development of GPUs. Our test setup S1 machine also contains the NVIDIA GPUs of TESLA and QUADRO family. Nvidia Tesla was named after pioneering electrical engineer **Nikola Tesla** and the TESLA series products targeted for stream processing or general-purpose graphics processing units (GPGPU). Nvidia Quadro is mainly for visualisation and graphics purposes, however, the same can also be utilised for general purpose computing via CUDA. The detailed investigation of computing performance is presented in the coming sections.

The Tesla C2075 GPU [36] used in this work consists of an array of 14 multiprocessors called the Streaming Multiprocessors (SMs). Each SM contains 32 Scalar Processors (SPs) cores therefore total $14 \times 32 = 448$ cores in a single GPU card. All SPs within each SM

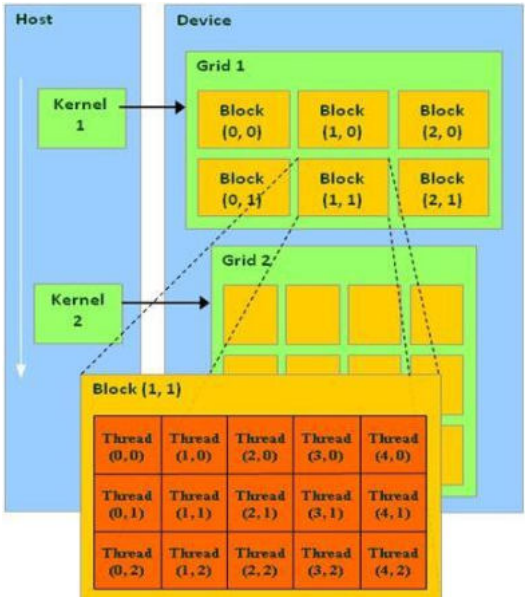

Figure 3.3: Grid, block and thread architecture of a GPU

share resources such as registers and memory. The instruction issue unit issues the same instruction across SPs inside a SM. Thus, execute the same instruction at any time like as SIMT. Figure 3.3 [38] shows a diagrammatic representation of GPU multiple grid, block, thread architecture. The thread hierarchy is of two levels. At the top-most level, there exists a two-dimensional grid of thread blocks. At the second level, the thread blocks are organized as a 3 dimensional array of threads. Kernel execution takes place in the form of a batch of threads organized as a grid of thread blocks. The thread blocks are scheduled across SMs. Each block comprises of many warps of 32 threads. Threads belonging to the same warp execute the same instruction over different data. The efficiency of computation is the best when the threads follow the same execution path for majority of the computation. Execution divergence, when threads of a warp follow different execution paths, is handled automatically inside the hardware. The size of the thread blocks (number of threads per block) and number of blocks can be managed by the program, and both values need to pass during the calling the GPU kernel.

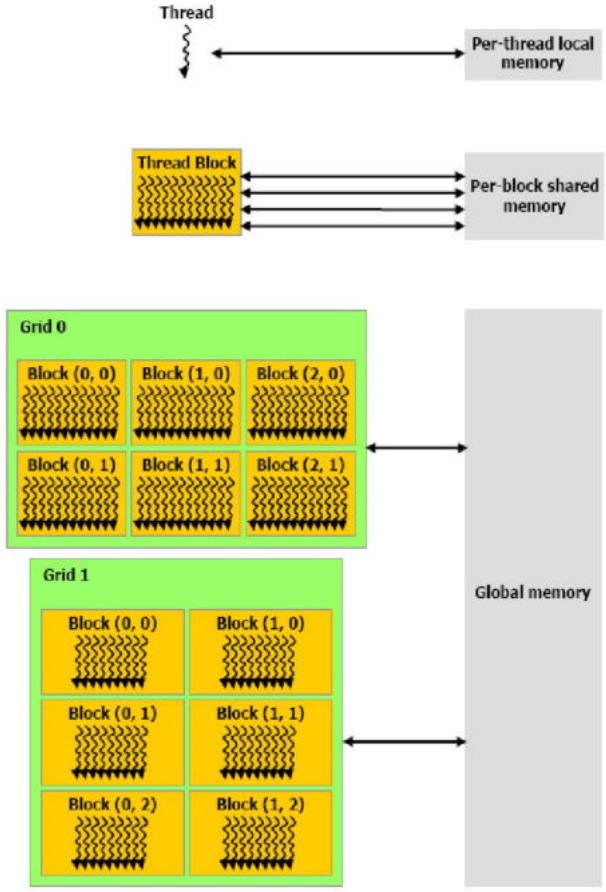

Figure 3.4: GPU memory organization

3.2.3 NVIDIA GPU memory architecture

In general, NVIDIA GPU's memory organisation is hierarchical as shown in Fig. 3.4. Each thread has its own private local memory. The Tesla C2075 GPU, includes a configurable L1 cache per Streaming Multiprocessor (SM) block and a unified L2 cache for all the processor cores. All threads inside a block share a memory space and this memory space local to thread is different for each block. Size of this local shared memory is 48 KB for the C2075 GPU. Life time of this memory equals that of the block and it is characterised by low memory access time. Shared memory comprises of a sequence of 32 bit words called "banks". There also exists a global memory shared across all the threads across all the thread blocks. The lifetime of this global memory is of the entire application. The size of this global memory is 6 GB for the C2075 GPU. Access time for this global memory is larger than the other

memories. Global Memory is composed of 128 byte segment sequence and memory requests for 16 threads (equal to a half warp) are serviced together. Each segment corresponds to a memory transaction. If the threads in a half warp access data spread across different memory segments (un-coalesced memory request), the corresponding multiple memory transactions would lower the performance. Our first implementation of the Brute-Force event selection algorithm on GPU could not achieve required performance due to un-coalesced memory access, the same is shown and explained below.

3.2.4 Implementation and optimisation of the Brute-Force event selection algorithm using CUDA

The Brute-Force event selection algorithm, described in Ch. 2, Sec. 2.4 has been implemented in C language using CUDA API and then compiled with the NVIDIA compiler (nvcc v7.0.27). Optimisation of the event selection code on GPU is not straight forward, therefore according to GPU architecture as described earlier, we analysed the Brute-Force event selection program and modified the memory arrangement accordingly [39]. Multiple events are processed at the same time, one event being allocated to one thread. After development and implementation of the event selection process using CUDA, we concentrated on optimising the CPU to GPU data transfer time, which is significant as the data volume is large. Data transfer time from CPU to GPU is an overhead but it is required towards utilising GPU computing capabilities.

Implementation and optimisation (i1):

In our first approach for implementation of event selection process, all the event hit data of the setup, consisting of the STS and MuCh detectors, were transferred to the GPU. The following steps were taken:

1. In the program data are read in from of a file. In the actual experiment, the data will be deployed in shared memory by the data acquisition software [9] and access to the

shared memory will be performed in parallel to the algorithmic computation. Note: The file input time is not included in all the execution time measurements presented in this thesis.

2. Memory on the GPU device is allocated for a chosen number of events n_{ev}(cudaMalloc).

3. Single precision hit data for n_{ev} events are transferred from CPU to GPU (cudaMemcpy).

4. The number of blocks b and the number of threads per block t are selected to optimise the GPU computation time.

5. The event selection algorithm is executed in GPU threads for n_{ev} events in parallel, with $n_{\text{ev}} \leq b \cdot t$.

6. The list of selected events is transferred back to the CPU (cudaMemcpy).

The GPU schedules and balances the selected number of blocks and threads on the available SMs (14 for C2075) and SPs (32). For processing, the data container with the hit coordinate array is arranged as

x_{11}, y_{11}, z_{11}, x_{12}, y_{12}, z_{12}, ... x_{1n}, y_{1n}, z_{1n}

x_{21}, y_{21}, z_{21}, x_{22}, y_{22}, z_{22}, ... x_{2n}, y_{2n}, z_{2n}

.....

x_{m1}, y_{m1}, z_{m1}, x_{m2}, y_{m2}, z_{m2}, ... x_{mn}, y_{mn}, z_{mn}

where x, y, z are the hit coordinates in configuration space; the first index denotes the event, the second one the consecutive number of the hit in the event.

Implementation and optimisation (i2) : Using coalesced memory access

Our investigation showed that this data arrangement suffers from uncoalesced memory access to the x, y, z coordinates (see also [39]). By its SIMT architecture, CUDA executes 32 threads of a block simultaneously; therefore all 32 threads should read from the global

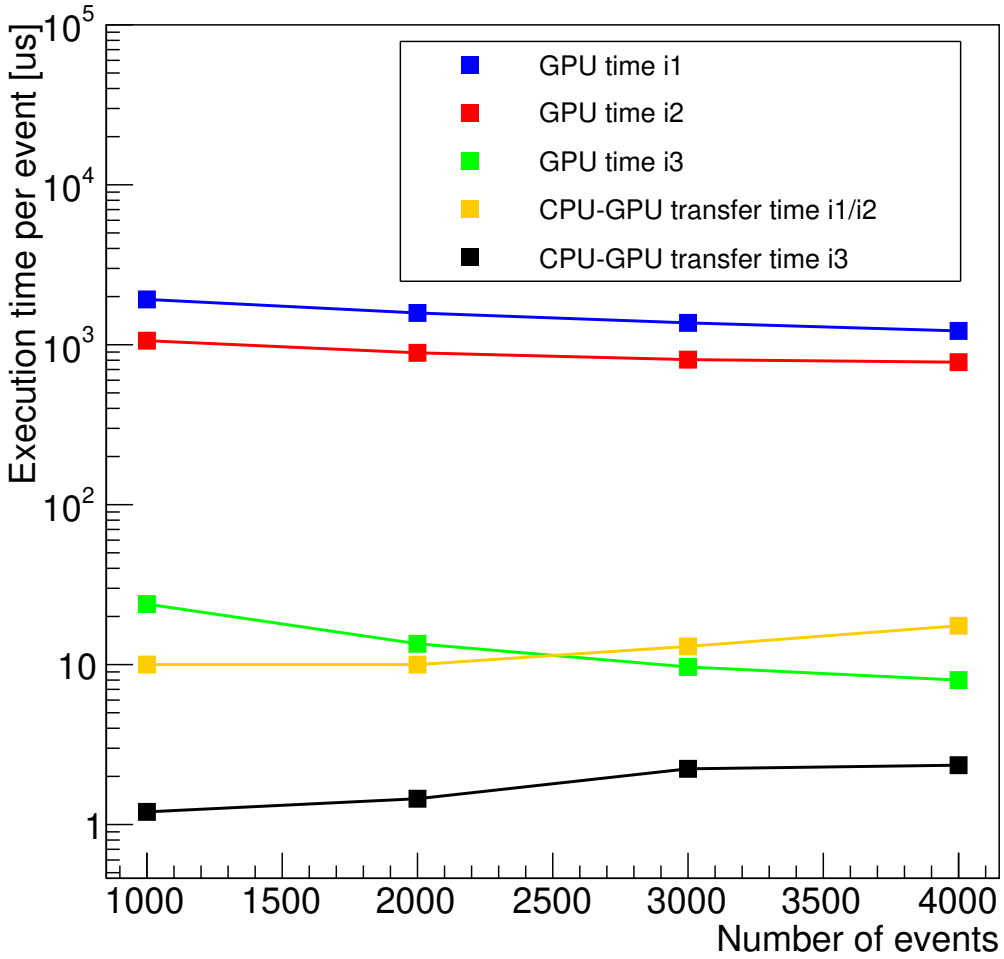

Figure 3.5: Processing time per event in microseconds as function of the number of events. The left panel compares the implementations i1, i2 and i3 with CUDA on the Tesla GPU (see text).

memory in a single or double read instruction. To cope with this, we rearranged the data such that coalesced memory access is possible. First, we introduced separate data containers for each coordinate axis, and second, hits of different events are arranged together:

x_{11}, x_{21},.....x_{m1}, x_{12}, x_{22}, ...x_{m2}.....x_{1n}, x_{2n}.....x_{mn}

y_{11}, y_{21},.....y_{m1}, y_{12}, y_{22}, ...y_{m2}.....y_{1n}, y_{2n}.....y_{mn}

z_{11}, z_{21},.....z_{m1}, z_{12}, z_{22}, ...z_{m2}.....z_{1n}, z_{2n}.....z_{mn}

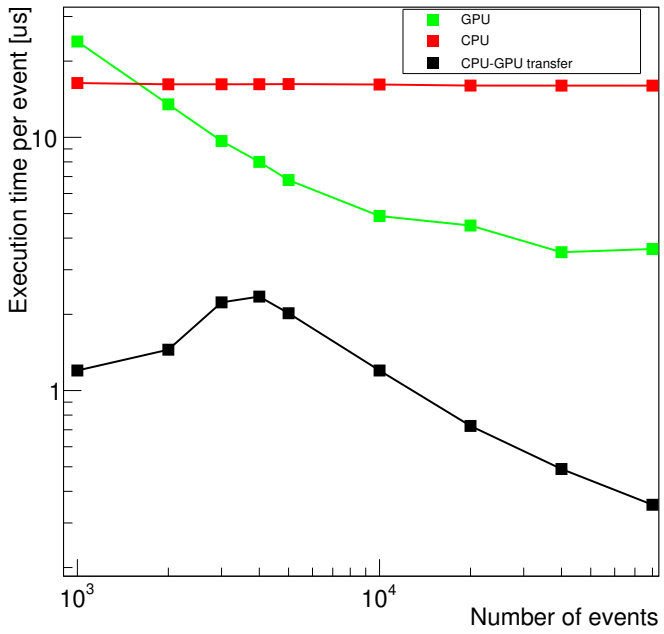

Figure 3.6: Processing time per event in microseconds as function of the number of events. A comparison of the execution times on GPU (implementation i3) and on CPU (single-thread) is shown.

Implementation and optimisation (i3): Using reduction in global read

In the course of further optimising the process, we found that the majority of time is taken by the global read of data by each thread, thereby requiring a reduction in global read time as data reside in the global memory and not in the shared or private memory of the GPU. By construction of the algorithm, the number of global reads for each thread is proportional to the number of events n_{ev}. Each event contains about 5000 hits, and every global read takes around 300–400 clock cycles [40]. For the computation, however, only a small fraction of these data are used, namely hits in the last (trigger) station, which are about 15 per layer per event (see Fig. 2.2). Thus, we introduced a filtering of the data on the CPU host side, such that only hit data in the trigger station are transferred to GPU [22].

The importance of the optimisation steps is illustrated in Fig. 3.5, showing the per-event GPU execution time for the various implementations on the Tesla GPU and respectively the per-event CPU to GPU data transfer time. This study was performed for up to 4,000

events because of memory limitations of the GPU. The processing time is reduced by a factor of two from the first implementation (i1) to the one properly using coalesced memory (i2). The data transfer time is the same for both implementations since the same data are transferred. Filtering of input data at the host side (i3) gives a reduction by about two orders of magnitude for the per event-execution time compared to (i2) and one order of magnitude for the per-event data transfer time.

Table 3.1: Results for the event selection algorithm on the Tesla GPU

# Events	# blocks	# threads	GPU Time (ms)	CPU-GPU Transfer Time(ms)	CPU Time (ms)	Speed-Up (CPU time/GPU time)
1000	32	32	23.9	1.2	16.38	0.69
2000	64	32	27	2.9	32.40	1.20
3000	64	64	29	6.7	48.60	1.68
4000	64	64	32	9.4	64.83	2.03
5000	128	64	33.9	10.1	81.16	2.39
10000	128	128	48.9	12	161.67	3.31
20000	256	128	89.7	14.5	320.24	3.57
40000	512	128	140.7	19.7	640.25	4.55
80000	1024	128	289.8	28.3	1280.39	4.42

Figure 3.6 and Table 3.1 compare the per-event GPU execution time and data transfer time for implementation i3 to the single-threaded execution time on CPU. The data filtering on the host side relaxes the restrictions imposed by the limited GPU memory, such that a larger number of events (we tested up to 80,000) can be processed at a time. The data transfer time is lower by one order of magnitude compared to the execution time; moreover, it can be hidden by performing computation and transfer in parallel [41]. The measurements demonstrate the importance to load the GPU with sufficient data in order to make optimal use of its capacity. Compared to the single-threaded execution on the CPU (using optimisation in the gcc compiler, discussed later), we obtain a speed-up of 4.55 for a data set of 40k events by using the Tesla GPU. Table 3.1 shows that for more than 40k events, the speed-up with respect to the single-threaded CPU is slightly reduced, indicating that

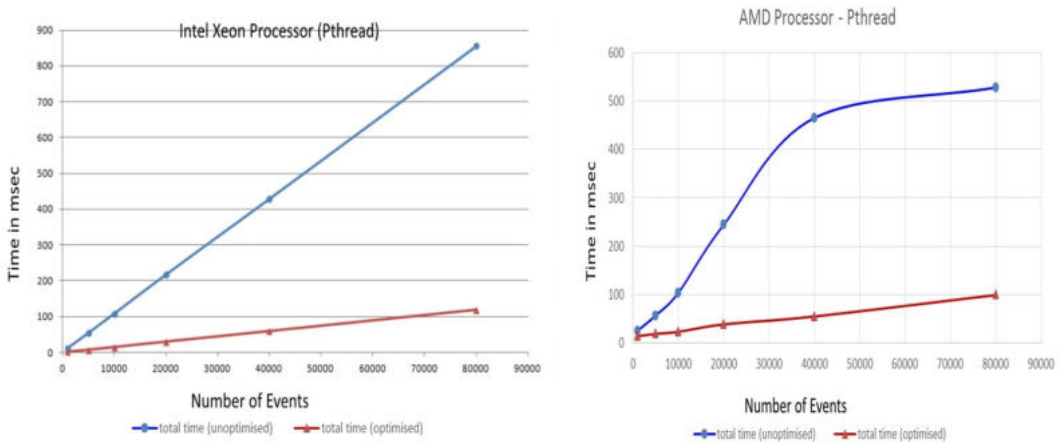

Figure 3.7: Comparison of the execution time between optimised with `-O2` option and unoptimised with `-O0` option, left: for Intel CPU and right: for AMD CPU.

the optimal data load on the GPU is reached with this amount of events. Our investigations show that about $3 \cdot 10^5$ events per second can be processed on a single Tesla GPU.

Optimisation on CPU using the compiler options:

After achieving optimal computing performance on the GPU, we started to investigate performance on the CPUs. GCC, **G**NU **C**ompiler **C**ollections, has been used to compile CPU programs written in C and C++ languages. Compiling program always has challenge of balancing between program performance, memory usage, compilation time, debugging information, etc. Other than interpreting the code and converting into a machine language that the system understands, gcc compiler also provides code optimisation options which can be set via `-O` flag. In particular, to optimise the execution time performance of any program, four options `-O0, -O1, -O2, -O3` are available with gcc compiler. Most optimisations are completely disabled at `-O0` and it is default option. `-O1` option tries to reduce code size and execution time, without performing any optimisations for compilation time. `-O2` performs nearly all supported optimisations that do not involve a space-speed trade-off. `-O3` option optimises more than `-O2`, however, execution time performance using these different options varies depending on the underlying program. It is not always the case that `-O3` flag optimises

the best with respect to the program execution time.

All these different `gcc` compiler optimisation options `(-O0, -O1, -O2, -O3)` have been investigated in detail for the CPU implementation of the Brute-Force algorithm and option `-O2` found to be optimal in context of total execution time. Figure 3.7 compares the execution time between optimised with `-O2` flag and unoptimised with `-O0` flag [42]. The left panel of Fig. 3.7 shows the comparison for Intel CPUs of S1 setup and the right panel shows for AMD CPUs of S2 setup.

All the execution time measured for CPUs in this thesis have been achieved after optimisations using the `-O2` `gcc` compiler flag. Also architecture specific option `-march = native` is used.

3.3 OpenCL: Open computing Language

OpenCL is a low-level API for heterogeneous computing and it provides parallel computing paradigm. It also provides a framework for writing programs consisting of CPUs, GPUs, digital signal processors (DSPs), filed-programmable gate arrays (FPGAs) and other processors. It includes a language for writing kernels and also works like an application programming interface (API) that is used to define and control the platforms.

3.3.1 Implementation with OpenCL and comparison with CUDA

The previous section has demonstrated that making optimal use of a GPU with the CUDA API is far from trivial and requires sophisticated optimisation of the data arrangement. The OpenCL programming paradigm [33] offers an architecture-independent alternative. However, for a beginner, OpenCL seems difficult as far as its syntax and programming procedures are concerned. Writing a small "Hello World" program in OpenCL needs creating platform, device, context, and command queue, then memory allocation via create and writing buffer, program object creation via creating source and building program, program

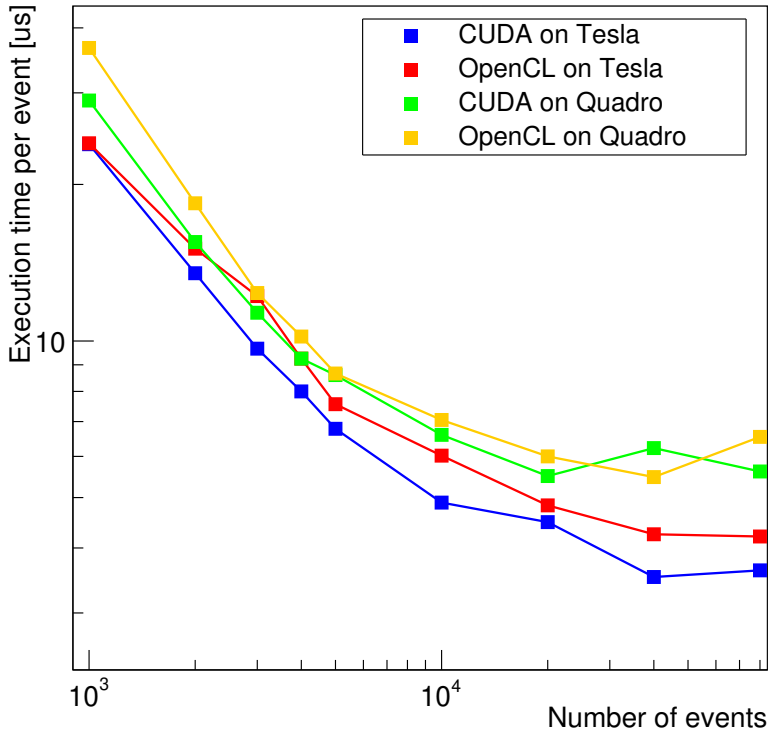

Figure 3.8: Execution time per event for CUDA and OpenCL implementations on the NVIDIA Tesla and Quadro GPUs

execution via create kernel, enqueueing kernel, reading back buffer, etc. At first sight, this seems cumbersome compared to CUDA which provides an easy terminology for writing programs [35]. On the other hand, OpenCL programs can be compiled via available C or C++ compilers, unlike CUDA which requires a vendor-specific compiler.

Once OpenCL programs are written and compiled, then can be executed on any device, whether GPU, CPU, or APU, whereas CUDA can be executed only on NVIDIA GPUs. Both CUDA and OpenCL treat the CPU as host, but for CUDA only the NVIDIA GPU is a device, whereas OpenCL treats any hardware as computing device by creating an instruction queue that can be executed on all available computing resources. We thus investigated OpenCL as an open-source solution for heterogeneous programming in the spirit of the studies presented in [43] and [44].

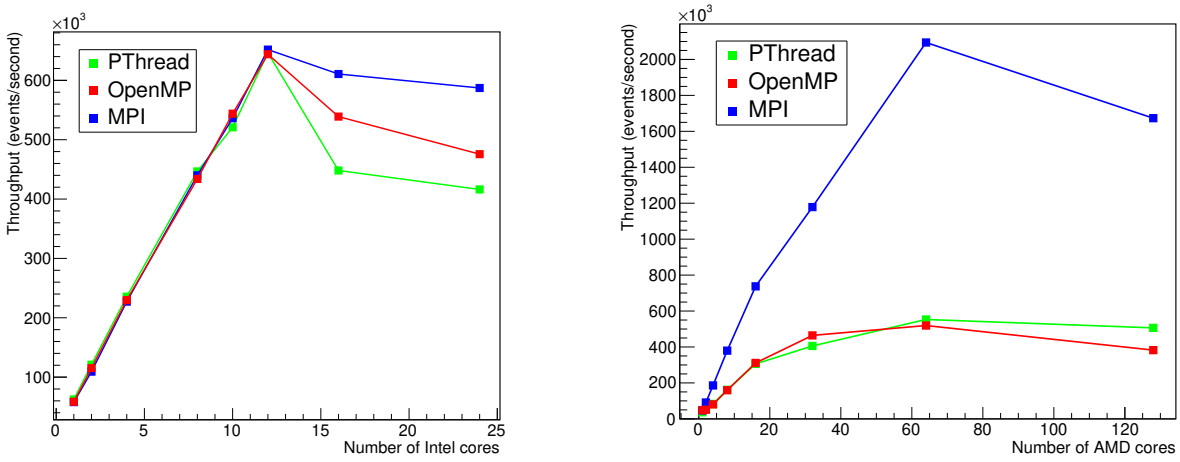

Figure 3.9: Throughput (number of events executed per second) obtained with the pthread, OpenMP and the MPI implementations as a function of the number of cores used in parallel for a sample of 20k events. The left panel shows the results for setup S1 (Intel, 12 physical cores), the right panel those for setup S2 (AMD, 64 physical cores).

Figure 3.8 shows the per-event execution time of the event selection algorithm for different numbers of events on the Tesla and Quadro GPUs using the implementations in CUDA and in OpenCL. We find the OpenCL code execution time to be slightly higher than that of the CUDA code on both Tesla and Quadro, possibly indicating that CUDA is better optimised to the NVIDIA GPU architectures. However, the difference is modest and seems a reasonable price for the flexibility offered by an architecture-independent code. Comparing Tesla and Quadro, we find the Tesla GPU to be more powerful, which becomes visible at large-enough input data (number of events). The hardware differences between these two GPU cards are manifold – processor speed, global memory size, number of computing cores etc [36, 37]. We conclude that the Tesla GPU seems more appropriate for our problem than the Quadro GPU.

3.4 Investigations on multi-core CPU

An alternative to using GPU accelerator co-processors is to make use of the multi-core CPU architecture present in contemporary computers [45, 46]. Concurrency on CPU cores can be established using pthread, OpenMP, MPI, and OpenCL, all of which are open-source programming paradigms, where OpenCL is primarily developed for many-core or GPU architecture. A preliminary study using pthread and OpenMP only was presented in [42], demonstrating the importance of the proper choice of compiler options. Here, we study in addition MPI and, in particular, OpenCL. We tested implementations of the event selection algorithm for all four of these programming paradigms on the two platforms S1 and S2; the GPUs of the S1 setup were idle or in open condition. Hardware parallelism was exploited in the simplest way by processing one event per thread (see Sec. 3.2.4).

Figure 3.9 (left) compares the throughput (number of events executed per second) on the Intel Xeon processors (2 x 6 cores) of setup S1 in dependence of the number of cores (threads) used in parallel for a sample of 20000 events. We find for all three pure parallel programming implementations a linear scaling with the number of threads up to 12 threads, from when on the throughput decreases again. This signals that from this point onwards, the context switching time starts to dominate the total processing time. The same test was performed on the setup S2 (4 x AMD Opteron 16 cores) as shown in the right panel of Fig. 3.9, obtaining similar results for 64 threads. OpenCL treats the underlying device, in this case the CPU, as a single compute unit; therefore, different timing results cannot be gathered by varying the number of cores.

For both setups, we find the throughput to scale with the number of threads / physical cores (the speed-up is 35 for AMD and 11 for Intel), which is to be expected for pure data-level parallelism. On the Intel setup, the performance obtained with pthread, OpenMP and MPI are similar, where MPI shows slight higher throughput. On the AMD setup, both pthread and OpenMP are less performant than MPI, although hardware-specific compiler flags were used. Note: thread spawning and distributing time are not accounted for the MPI

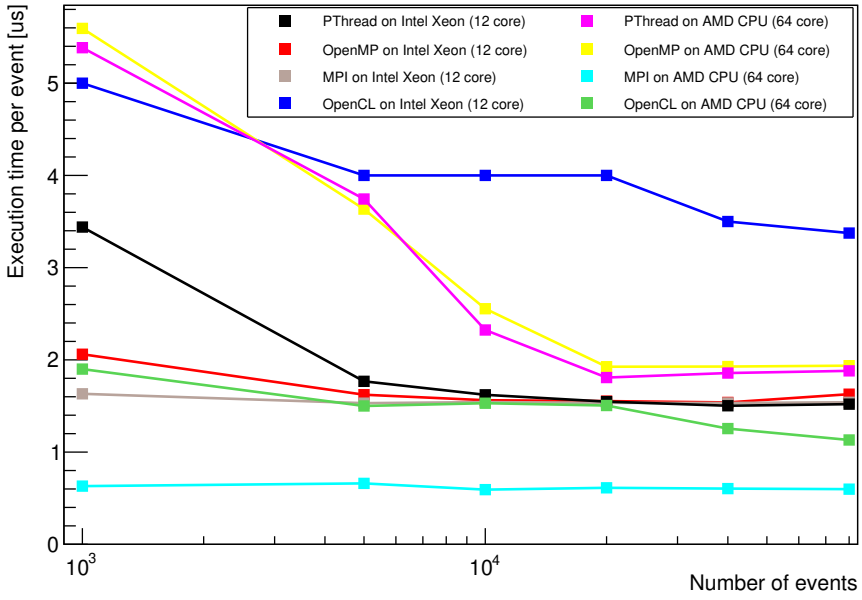

Figure 3.10: Execution time per event as function of the number of events processed at a time for the implementations with pthread, OpenMP, MPI and OpenCL on the Intel and AMD CPUs. The number of threads equals that of the available physical cores (12 for Intel, 64 for AMD).

implementation; they contribute in proportion to the number of threads. We attribute our findings to the fact that the event selection process does not use shared memory or inter-thread communication. Unlike the other frameworks, MPI statically binds the thread to CPU cores. The similarity of the results for the pthread and the OpenMP implementations are to be expected since internally, OpenMP uses pthread for spawning multiple threads.

The performances obtained with OpenCL, pthread, OpenMP and MPI on the two hardware architectures are compared in Fig. 3.10 for different numbers of events processed at a time (up to 80,000 events). The number of threads is 12 for Intel and 64 for AMD, as shown to be optimal by Fig. 3.9. On both platforms, we find the execution times for pthread and OpenMP to decrease with the number of events and then saturate, indicating a minimal data size (about 20k events) from which on the process overhead can be neglected. On Intel, OpenCL performs clearly worse than the other implementations, whereas it is slightly better

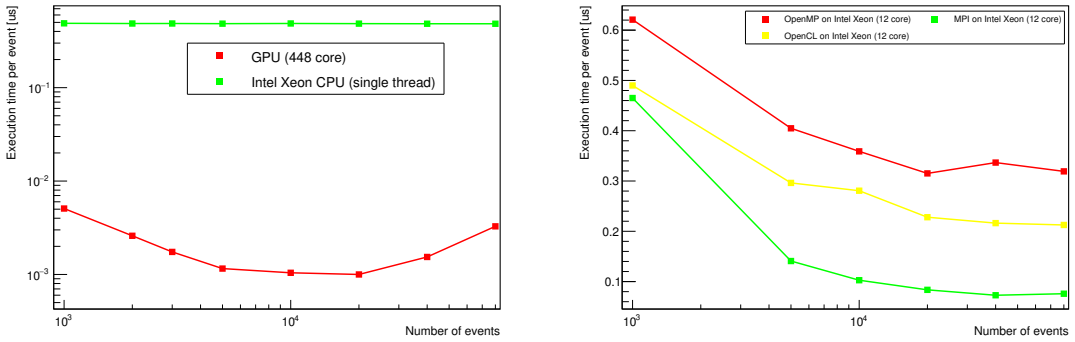

Figure 3.11: Execution time per event as function of the number of events processed at a time for the implementations (a) with GPU comparing to CPU (single-thread) (b) with OpenMP, MPI and OpenCL on the Intel CPU.

than OpenMP and pthread on AMD; here, MPI is found to clearly give the best execution speed. As reasons for OpenCL to perform worse than e.g., MPI, we have to acknowledge the fact that OpenCL was primarily designed as a GPU programming tool; thus, its performance is proportional to the number of thread invocations. OpenCL also produces vectorised code in an automatized way, whereas manual vectorisation and/or compiler optimisation flags need being used for better performance on other implementations [6].

A comparison of the Intel and AMD processors is not straight forward because of the differences in the number of computing units and theoretical peak performances. Considering only the throughput per core, AMD appears less performant than Intel for all implementations since the number of parallel threads is 5 times larger than for the Intel CPUs, but the per event execution time is only 2.5 times smaller. A complete assessment, however, would have to also take into account the costs for purchase and operation, which is beyond the scope of the thesis.

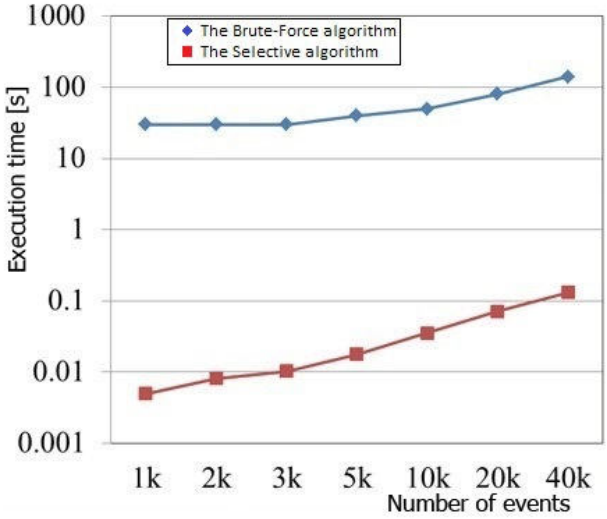

Figure 3.12: Comparison of execution time of event selection process on NVIDIA Tesla GPU with both the algorithms, the Brute-Force algorithm (blue) and the Selective algorithm (red).

3.5 The Selective algorithm results and comparison with the Brute-Force algorithm

Upto now all the studies presented in this chapter is for the Brute-Force event selection algorithm. Similar studies have also been performed for the Selective algorithm with the setup S1 machine. As described in Sec. 2.5, the algorithmic complexity is an order lesser than the Brute-Force algorithm complexity. The effect of algorithmic revision of the Brute-Force algorithm is clearly visible in all the execution time measurements. For completeness, the left panel of Fig. 3.11 shows per event execution time as a function of the number of events for the NVIDIA Tesla GPU and compare with the execution time on S1 setup CPU (single thread). The right panel of Fig. 3.11 shows the per event execution time for the OpenMP, MPI and OpenCL implementation of the Selective algorithm on the Intel CPUs of setup S1.

Figure 3.12 shows comparison of the execution time with varying number of events for both the algorithms. Note: the execution time is not normalised with number of events.

The algorithmic improvement of the Selective algorithm than the Brute-Force algorithm in terms of the execution time is clearly seen in the Fig. 3.12.

3.6 Conclusions

We have described the development of an event selection algorithm for the CBM-MuCh detector and a systematic study for the implementation of the event selection process using different parallel computing paradigms like pthread, OpenMP, MPI, and OpenCL for multi-core CPU architectures, and CUDA and OpenCL for many-core architectures like NVIDIA GPUs. For both the platforms, the event selection procedure suppresses the archival data rate by almost two orders of magnitude without reducing the signal efficiency, thus satisfying the CBM requirements for high-rate data taking.

On GPUs, we have found a speed-up of 4.5 with respect to the optimised single-thread execution on CPU. This result, however, is only obtained after careful optimisation of the implementation in CUDA. OpenCL on NVIDIA GPUs are found to perform only slightly worse than that for CUDA. Our results show that about $3 \cdot 10^5$ events per second can be processed on a single GPU card of NVIDIA Tesla family. Present hardware supports up to four GPUs on a single motherboard. This suggests that the targeted CBM interaction rate of 10^7 events per second can be accommodated by a small number of servers properly equipped with GPUs.

In a multi-core CPU environment, we have compared OpenCL, pthread, OpenMP and MPI as open-source concurrency paradigms. A linear scaling of the data throughput with the number of parallel threads is observed up to the number of available physical cores. In the powerful S2 setup with in total 64 AMD cores, we find that about $2 \cdot 10^6$ events can be processed per second, which is already close to the targeted event rate of 10^7/s. This demonstrates that SIMD instructions provided by modern CPUs are essential to achieve the required throughput, and that the computing demands of the CBM experiment for the

real-time selection of J/ψ candidate events can be achieved by properly making use of the parallel capacities of heterogeneous computing architectures. As an example, the NVIDIA Tesla GPU of setup S1 could be placed into setup S2 to achieve the desired goal.

Comparing the different programming paradigms, we find the cross-platform OpenCL to be a proper choice for heterogeneous computing environments typical for modern architectures, which combine CPU cores with GPU-like accelerator cards. For such kind of systems, OpenCL provides a suitable solution to simultaneously exploit all available compute units for a given application. It also provides the flexibility to future improvements in computing architectures, which is of particular importance for CBM as an experiment in the construction stage. This flexibility, however, comes at the price of a reduced performance on CPU when compared to pure parallel programming paradigms.

Chapter 4

Digitization and Time Based Simulation

4.1 Introduction

One of the major goals of the CBM experiment at FAIR is the investigation of rare probes. To achieve the goal, it is required to build the facility which can run with an unprecedented interaction rates and good statistics of rare probes can be collected. For the same to cope with the high interaction rates (∼10 MHz), self triggered electronics have been instrumented and every signal message on each readout channel will be gathered. In the real experiment, gathered information from the data acquisition system will be in the form of hardware messages which will be converted into subsystem or detector specific objects termed as "digi". Digi is the representation of single raw data or the smallest unit of raw data, in the context of the CBM software stack. In this respect, almost realistic digitized information is a must for performance simulation of the experiment.

Digitization is the key process for simulating the realistic output of a subsystem. The task of the digitization is to calculate the detector response to a track traversing an active detector element. The process mainly depends on the underlying detector technology and therefore each subsystem is liable to provide the realistic implementation. Before digitization, one needs to understand the basic of input and output for the digitizer for any HEP simulation

Figure 4.1: Schematic for event by event simulation flow.

process. Figure 4.1 depicts the simulation work flow for event by event simulation which is based on the event by event processing. A large fraction of the HEP experiments so far are based on the hardware-trigger approach, and therefore the entire software stack comprising blocks like simulation, reconstruction had been developed keeping generation of event at the hardware level in mind. As depicted, after the transport simulation, using a suitable transport engine (e.g. GEANT3), MCPoint objects have been created which represent particle interaction points inside the active medium of the detector. The MCPoint object, termed as `CbmMuchPoint` for MuCh, stores the geometrical information of the track intersection with the detector. This MCPoints Array and the related information have been stored event by event in the permanent storage in the form of a ROOT tree, in which one entry of the tree represents one event. In `CbmRoot`, it is represented as `CbmEvent` branch. This entire data flow works in synchronous mode and each individual process takes one event as input and produces one event as output and there is one to one correspondence. This architecture of simulation flow needs to be redesigned for the time based simulation. Figure 4.2 depicts the probable solution for generation of a free running timestamped data stream in the form of time slice and stored as `CbmTimeSlice` in a ROOT tree for further time

based simulation. Here multiple simulated MC events are taken into the digitizer (described later) and `CbmDaq` generates time-slice. At this point the one to one event association is destroyed and an asynchronous mode is adapted. These time slices will be equivalent to time stream data gathered from self triggered electronics and DAQ. From these time slices, after event building, `CbmEvent` will be created. In event-by-event and time-based, further reconstruction tasks operate on input vector of digis (described later) and produce output array, the vector and the array corresponds to one event or one timeslice, respectively. Figure 4.3, shows the further reconstruction of clusters, hits, and tracklets from digis.

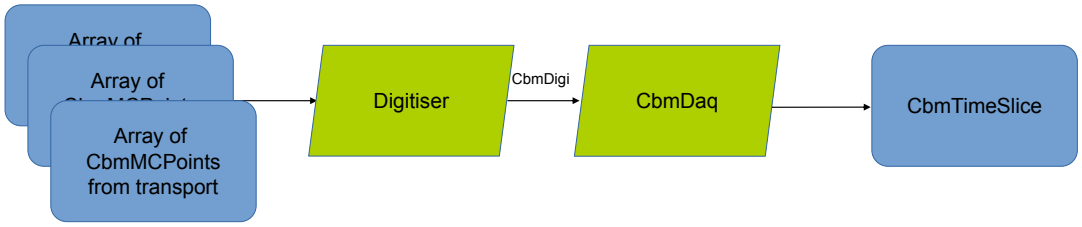

Figure 4.2: Schematic for time based simulation flow.

To have a realistic time based scenario, the MuCh digitizater should fulfil the following criteria,

(a) Implementation of a realistic detector response,

(b) Should work in both the event by event mode and the time-based (free-streaming) mode,

(c) Should be compatible with the framework scheme of the in-memory buffer,

(d) Proper treatment of the interference of tracks in a given readout channel both in the event mode (interference within one event) and in the time-based mode (interference within the dead time of the electronics, within or across events),

(e) Generation of time-based digis and then store them into time stream data such that it describes the self-triggered, free-streaming readout of the detector,

(f) Noise generation (originated from the electronics and the physical processes).

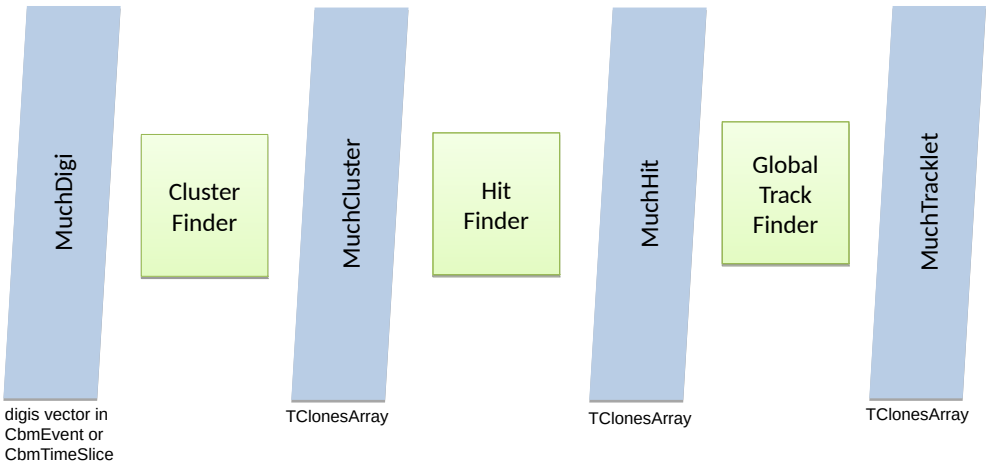

Figure 4.3: Schematic reconstruction data flow after digi creation.

In this chapter, we describe the entire digitization process in detail covering all the above requirements, with emphasis on realistic time stamping to every digi specific to the MuCh detector subsystem. This corresponds to creating the MuCh digi (`CbmMuchDigi`) objects from the Monte Carlo Point (`CbmMuchPoint`) objects.

Later in this chapter we will provide a detailed performance study which is relevant for the implementation of a trigger-less free-running data acquisition system. It describes the implementation of a free-streaming detector simulation for the muon detector using GEMs and RPCs and analyses the performance and the rate-dependent effects (e.g. data losses arising from pile-up) of the DAQ system.

4.2 Detector response simulation

4.2.1 Simulations in `CbmRoot`

In `CbmRoot`, simulations are performed in two separate steps with intermediate file output. In the first step called "transport simulation", the primary particles are traced through the detector, taking into account their trajectory in the magnetic field, their interaction with the materials and - for unstable particles - their decay. Secondary particles created by decay or by interaction with materials are created and traced as well. This step employs external transport engines; the framework makes use of the ROOT TVirtualMC [47] features which allow to choose between GEANT3 and GEANT4. The simulation results presented in this thesis are with the GEANT3 transport engine. At transport level, a realistic description of the detector geometries, the proper material properties and a map of the magnetic field are required [19]. As output of the transport simulation, the geometric intersections of particle trajectories with the active detector elements ("MCPoints") are recorded, along with the time-of-flight from the event start and the energy deposit in the active material. These information are stored in a file and provides the input for the second simulation step.

The task of the detector response simulation is to model the physics processes in the active detector material, the readout and the digitization in the front-end electronics. The output objects called "digi" represent the simulated measurements of a single read-out channel - corresponding to one read-out pad in case of the MuCh. This atomic data unit, "digi", contains the information on the respective channel address, the time of the measurement

termed as time-stamp, and the digitized charge. The digi objects are then forwarded to the software emulation of the CBM DAQ system as implemented in `CbmRoot`. At this step, the association of digis to events is destroyed as described in Sec. 4.1, resulting in a data stream similar to that expected from the actual experiment. In the time-based simulation, based on the dead-time discussed later, a single pad might be recorded as two digis with different time-stamps.

In the context of a free-streaming readout as described in Sec. 4.1, the correct description of the timing behaviour is of utmost importance. The effects contributing to the finally registered time stamp are:

- Event time: the time corresponding to the actual collision. It is generated by the `CbmRoot` framework from the time profile of the beam and the interaction probability of the beam particles in the target.
- Time-of-flight: from the event start to the intersection with the detector element. This time is provided by the transport engine and stored in the MCPoint object.
- Drift time: the time taken by the primary electrons in the GEM or RPC detectors in MuCh to drift to the readout planes.
- Time response of the readout ASIC to the analog charge collected in the readout pads.

An important figure is the dead time of the readout electronics, i.e., the time after a hit in which the corresponding readout channel is blocked, such that a second hit arriving within this time is neglected. This leads to a detector inefficiency and track pile-up which are obviously dependent on the interaction rate, i.e., the time separation between two subsequent collisions.

4.2.2 Analogue response simulation

The working principle of a multi-layer GEM detector is illustrated in Fig. 4.4. A traversing charged particle creates a number of primary electrons in the drift gap through ionisation

of the gas. The primary electrons drift in the electrical field towards the GEM foils, where amplification through the creation of avalanches takes place. This process is repeated in various transfer gaps between the successive GEM foils. Finally, the produced electrons are further amplified. Amplification takes place in the GEM holes. The drift and transfer fields are there to guide the secondary electrons to the next GEM foil. Fields in the induction gap helps in collecting the charge on the pads of the readout PCB.

Our analogue simulation implements a simplified scheme as shown in the Fig. 4.5. The trajectory of the charged particle in the drift gap is approximated by a straight line, the coordinates of the entry and exit points being provided from the transport simulation stage. For each particle, the number of primary electrons is sampled from a Landau distribution, the parameters of which depend on particle type, energy, track length in the active volume and specifications of the gas mixture [48]. The primary electrons are randomly generated along the trajectory in the active volume with a uniform distribution.

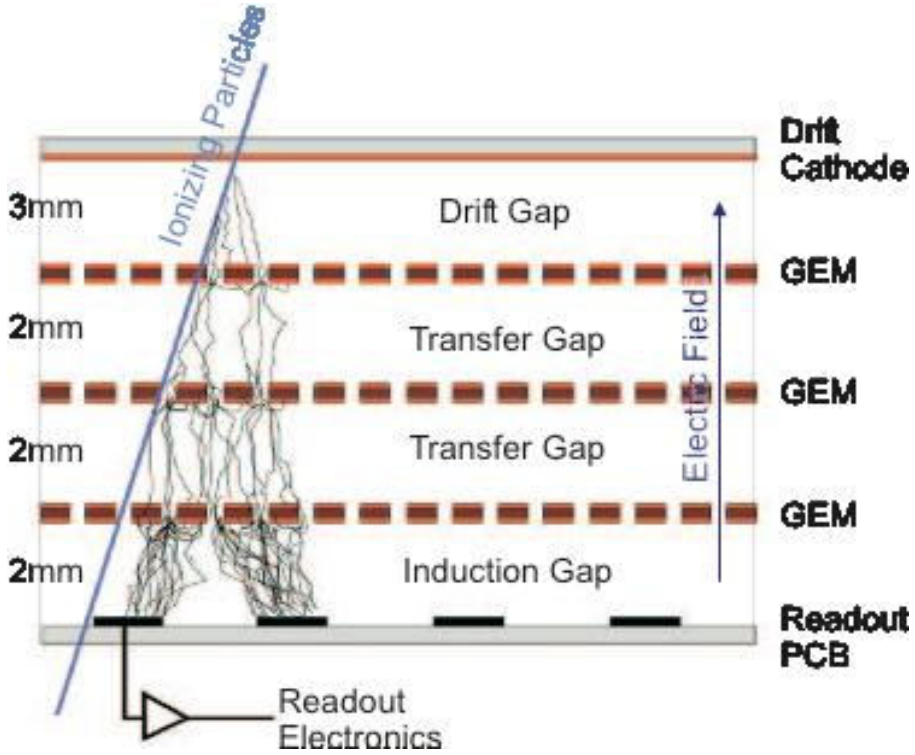

Figure 4.4: Working principle of a multi-layer GEM detector

For each primary electron arriving at the amplification gap, the number of secondary electrons is calculated from the gas gain settings, and the charge depositions to the gaps are calculated assuming a spot radius of the avalanche. Depending on the track inclination, the gas gain and the avalanche profile, a particle can activate one or several pads. The drift time is the time taken by the primary electron to traverse the active volume (up to the first GEM foil). The avalanche travels close to the speed of light, such that its propagation time is negligible in comparison to the drift time. Primary electrons generated in the passive volume do not contribute measurably to the total signal since no avalanche production takes place. Transfer gaps between the GEM foils and the induction gap between, the last GEM foil and readout PCB are combined and termed as passive volume as shown in the Fig. 4.5.

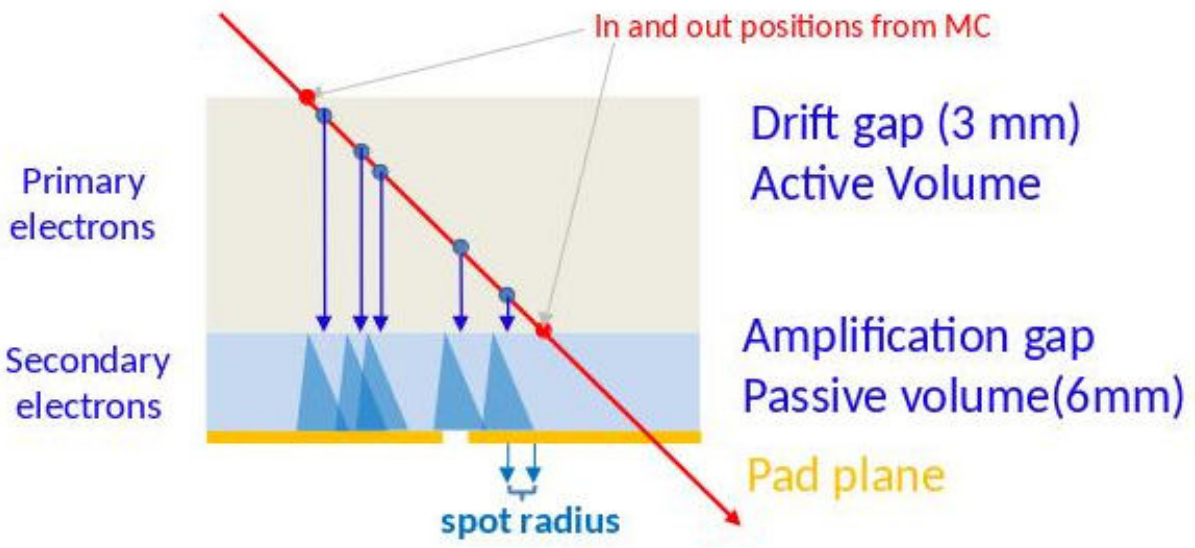

Figure 4.5: Simplified scheme of GEM detector as implemented in the analogue response simulation.

The parameter input to this analogue simulation consists of the parameters of the Landau distribution for the number of primary electrons, the drift velocity of the primary electrons, the gas gain, and the avalanche spot radius. All these parameters are tunable in the `CbmRoot` software. The size of the drift gap is taken as per geometry of GEM or RPC. The analogue

simulation process is the same for GEM and RPC detectors, but with different parameters. The default settings for both detector types are summarised in Table 4.1.

Table 4.1: Parameters for the analogue simulation of GEM and RPC detectors

Parameter	GEM	RPC
MPV for primary Landau distribution	from HEED*	12
Drift gap (fixed as per detector geometry)	3 mm	2 mm
Drift Velocity of primary electrons (μm/ns)	100	120
Gas gain	5000	30000
Spot Radius	500 μm	2 mm

* a code to generate primary electrons.

The analogue simulation as described above is performed separately for each input MC-Point. The resulting charge depositions to the readout pads, however, cannot directly be digitized, since they block the respective electronics channel for a certain amount of time (dead time), thus potentially influencing later measurements. Such incidences of different particles contributing charge to the same pad can happen within one event (conventional pile-up), but in our free-running scenario also for particles from different events, provided the difference in the event time is comparable or smaller than the dead time. To cope with this situation, the registered charges per pad are internally buffered and keeps a match object associating a link per corresponding incident track. Later this match information is used for finding the primary and secondary track contribution towards generation of digi or pile-up. The stored information also contains the pad address, the signal time, the charge, and the duration of the active signal.

4.3 Digital response simulation

The digital response to the accumulated charge in the readout pads is modelled following the properties of the readout ASIC, the STS/MUCH-XYTER (SMX) [49, 50]. This ASIC features two shaping channels, the fast shaper used to determine the time stamp of the

measurement, and the slow shaper measures the charge amplitude by time-over-threshold. Once a signal in the fast shaper crosses a pre-defined threshold, the signal in the slow shaper is evaluated and a message to the DAQ system is issued when the amplitude in the slow shaper falls below a second threshold. The signal shape in the slow shaper thus defines under which conditions two subsequent signals in the same readout channel can be disentangled. If two signals could not be disentangled then it is termed as pile-up.

The realistic functionality of merging of two signal shapes due to pile-up had been implemented (described below) and a new signal shape is obtained by bin-by-bin addition of the two merging signal shapes. A detailed study had been performed regarding charge deposition effect due to the pile-up cases. In the process, each primary electron generated during particle tracklet passing via the active volume of the detector had been taken and according to gas gain and spot radius resulted charge spot area was created. A `CbmMuchSignal` is created for each pad which falls under the area created by spot radius. Then this intermediate signal is stored in the buffer. The same process is repeated for all the primary electrons for the same tracklet and new `CbmMuchSignal` objects are created. Subsequently during storing in the intermediate buffer (described later), if this new signal interfere with older signal in the buffer, it is detected as pile-up and the corresponding `CbmMuchSignal` is modified with bin by bin addition of signal shape accordingly. This process is then performed for each primary electron of every MC Point. This realistic implementation has been committed in `CbmRoot` version of 2016 [51]. The described realistic process of electronics chip consumes huge system memory and processing time. This process is realistic but output of SMX chip is only 5 bit ADC value corresponding to the accumulated charge, therefore, it is decided that this high compute intensive calculations are not required [52], the same has therefore been replaced with a rather simple version as described in the next paragraph.

Now merging of two signals is modelled in the latest version of `CbmRoot` software by a delta function for the signal shape, the start time being the time stamp and the duration being the dead time of the ASIC [53]. The amplitude is the total charge. Two overlapping

signals (i.e., the second one arriving within the dead time of the first) are merged into one, having the start time of the first signal, the stop time of the second, and the amplitude is the sum of both the signals. The generated signals are transiently stored in a readout buffer. When the system time (the time of the currently processed event) exceeds the stop time of the signal, it is digitized, since then it is guaranteed that it will not be influenced by subsequently arriving signals.

All analogue signals exceeding the threshold are digitized, using the parameters of the ADC built into the SMX. The charge is linearly discretised, above a tunable threshold, into 32 bins corresponding to the 5 bit resolution. The time of the digital signal is smeared by a Gaussian distribution representing the time resolution of the ASIC. The parameter input to the digital response simulation are thus the number of ADC channels, the ADC threshold, dynamic range, and the time resolution. The ADC dynamic range is defined as `fQMax-fQMin` in the MuCh Digitizer. The currently used values of these parameters are summarised in Table 4.2. It should be noted that these parameters are still subject to changes; in particular, the dead time, which is a critical parameter for the performance of the detector system, is expected to be smaller for SMX v2.2 currently under development [54].

Table 4.2: Parameters for the digital response simulation of GEM and RPC detectors

Parameter	GEM	RPC
Number of ADC channels	32 (5 bit)	32 (5 bit)
ADC threshold (Min charge equivalent in fc)	2 fC	30 fC
Maximum charge (equivalent in fc)	80 fC	130 fC
ADC dynamic range (`fQMax-fQMin`)	78 fC	100 fC
Dead time	400 ns	400 ns
Time resolution	5 ns	5 ns

4.3.1 Implementation in `CbmRoot`

In the context of the `CbmRoot` framework, MuCh digitization is implemented as `CbmMuchDigitizeGem` task class and the same is compatible with the `FairRoot` framework which is instan-

tiated using `FairRunSim` or `FairRunAna` kind of steering classes. Earlier [51] this task class was compatible for event mode simulation only. It took a long time to convert the existing classes from the event mode to a time based mode. Each digi contains a unique 32 bit `CbmMuchAddress` and for each detector channel `CbmMuchPad` is assigned with one `CbmMuchAddress`. Each `CbmMuchDigi` is associated with `CbmMuchPad` uniquely using `CbmMuchAddress`. Here `CbmMuchPad` is active if a particular detector element is fired and the corresponding digi is created.

For free streaming scenario or time based mode, track interference can happen within an event and also across the event. To implement generation of time stamps for each digi, we investigated all the relevant classes. As discussed earlier, that `CbmMuchDigi` is associated with `CbmMuchAddress` and in turn with `CbmMuchPad`. The `CbmMuchPad` is the first choice to implement timing measurement. During simulation in `CbmRoot`, all the pads are reset after each event, therefore to keep active channel information in the `CbmMuchPad` is not feasible. A detector channel, `CbmMuchPad`, can also be fired due to inter event track interference for the free streaming scenario and `CbmMuchPad` is reset after each event. To keep some information across the events, a buffer is required, which is implemented as `CbmReadoutBuffer` in the `CbmRoot` framework.

For time based implementation, MuCh related data classes must be compatible with the following requirements,

1. to work both in event mode and time-based (free-streaming) mode;
2. be compatible with the framework scheme of `CbmReadoutBuffer`;
3. properly treat interference of tracks in a given readout channel both in the event mode (interference within one event) and in the time-based mode (interference within the dead time of the electronics, within or across events).

To cope up with the requirement of implementation of interference a new class `CbmMuchSignal` has been introduced to keep the signal shape according to the energy deposition of the in-

cident track. For each channel/pad, a `CbmMuchSignal` is generated, which describes the analogue response. It contains the unique detector address, the signal time, the time until the signal is active and can be influenced by subsequent signals (stop time), and the signal shape represented by a `TArrayD` in steps of nanoseconds according to the charge accumulation formula [49].

The time of the signal, which is crucial for a correct description of the free-streaming behaviour, is calculated from the event start time (obtained from `FairRunAna`), the time of the `CbmMuchPoint` (time-of-flight from event start to detector) and the drift time in the GEM/RPC.

The created signals are buffered in the `CbmMuchReadoutBuffer` singleton object, deriving from the `CbmReadoutBuffer` template. This buffer is responsible for dealing with pile-up. In case a second signal arrives in a given readout channel (pad) within its dead time, the `Modify()` function is called. This method merges the two signals into one. The start time of the resulting signal is the minimum of the start times of the merged signals, the stop time is the maximum of the two stop times. The new signal's amplitude is obtained by the sum of both the signals.

After processing all `CbmMuchPoint` objects of one event in this manner, the readout buffer releases all buffered signals with a stop time before the current event time, since they cannot be influenced by following signals any more. In case of event-by-event simulation, the entire buffer is read out irrespective of time, which excludes interference of tracks from different events, however, interference of tracks within the single event has been taken care of.

`CbmMuchSignal` objects released by the readout buffer are converted into digital information (`CbmMuchDigi`) taking into account the properties of the readout ASIC like number of ADC bins, dynamic range and threshold. As described earlier, STS-MUCH-XYTER chip [54] is used for the readout of 128 readout channels. Detailed description and working principle of the SMX chip is not in the scope of this thesis, however, timing features, ADC,

dynamic range etc of the chip have been studied and considered for the precise and realistic implementation.

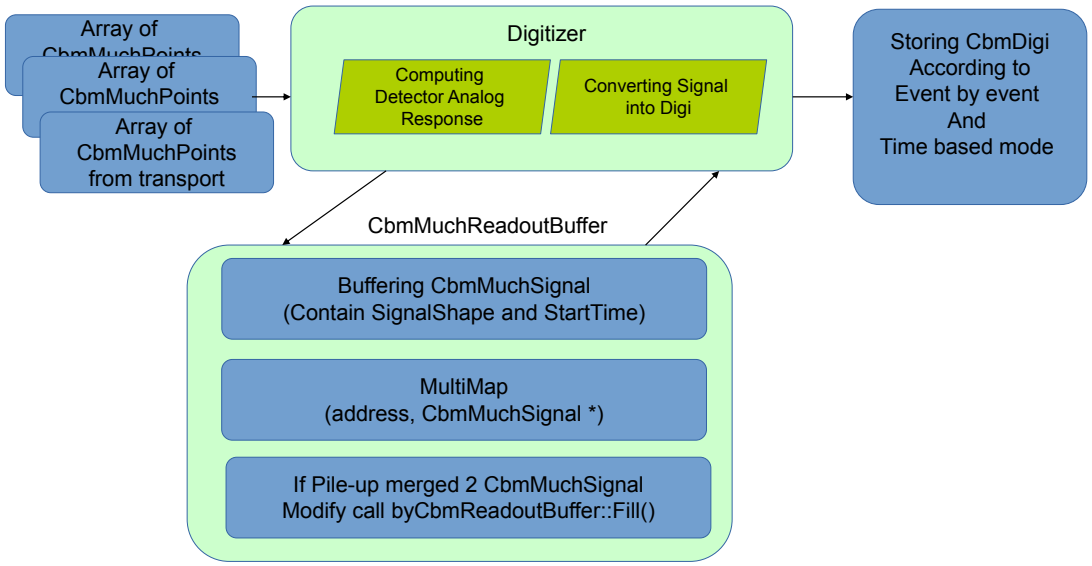

Figure 4.6: Logical diagram for MuCh digitizer for time based, signal buffered scheme.

The digis (`CbmMuchDigi`) are delivered to the `CbmDaq` software instance, which aggregates digis from all detector systems, builds time slices and stores them in the output tree. In case of event-by-event simulation, the digis are directly written into the output tree. The work flow for digitization in MuCh is schematically shown in Fig. 4.6.

4.3.2 Dead-time of Electronics

An important figure is the dead time of the readout electronics, i.e., the time after a hit in which the corresponding readout channel is blocked, such that a second hit arriving within this time is neglected. This leads to a detector inefficiency which is obviously dependent on the interaction rate, i.e., the time separation between two subsequent collisions. In case a second signal arrives in a given readout channel (pad) within its dead time, the buffered signal is modified and the two signals are merged into one. The start time of the resulting signal is the minimum of the start times of the merged signals, the stop time is

the maximum of the two stop times. Typical signal shape and general electronics response is shown in Fig. 4.7. Implementation of realistic deadtime is under continuous development as the SMX chip has undergone major revisions and SMX v2.2 [54] will be released in near future using fast response (90 ns configuration) of the slow shaper. The SMX chip has 4 different configurations which vary from 90 - 280 ns [54].

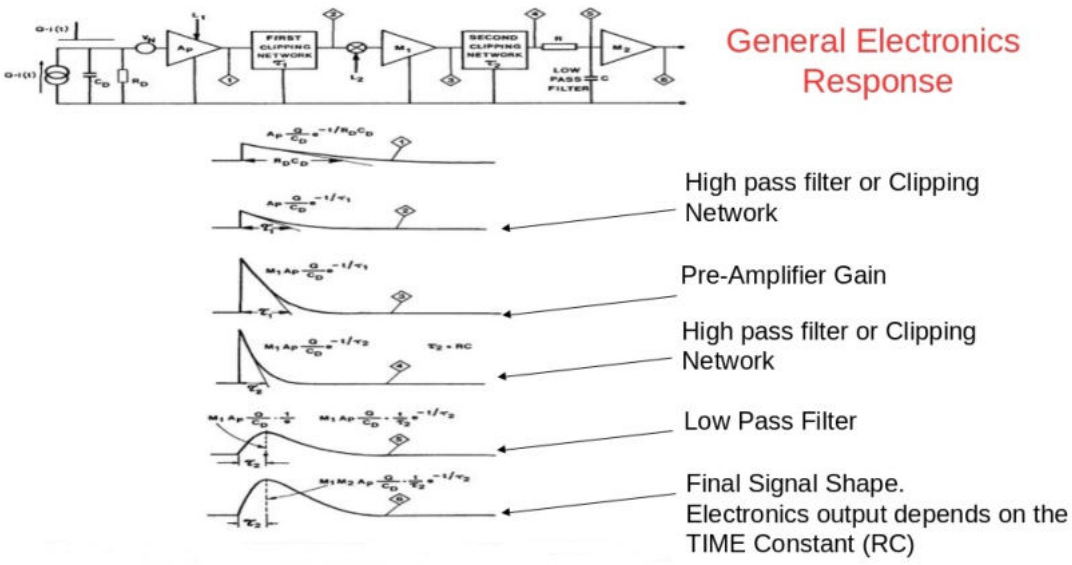

Figure 4.7: General electronics response and signal shape generation.

Signal objects released by the readout buffer are converted into digital information (digi object) taking into account the properties of the readout ASIC. SMX ASIC [54] is used for the readout of 128 channels and each channel is uniquely connected with a single pad of the detector by almost maintaining the same track length for each channel.

In a nutshell, the task of the MuCh digitization is to calculate the detector response to a track traversing an active detector element. Inside `CbmRoot`, this means creating `CbmMuchDigi` objects (representing the smallest unit of raw data) from `CbmMuchPoint` objects, which store the geometrical information of the track intersection with the detector obtained from transport simulation. To save the run-time memory, drift-time calculation is performed only once for a track or a MC Point, not for all primary electrons generated

during drift gap for a single track.

Detailed pile-up effect investigation is performed and described in coming Sec. 4.4. It is seen that for the MuCh setup, pile-up at high interaction rate has a significant impact on the first station.

4.3.3 Noise generation and simulation

There are two types of noise, (a) Electronics Noise and (b) Physics Noise. Both are not event correlated noise (not correlated to any particle produced in the event) and makes more sense in the time-stream mode, which will be part of the real data.

Electronics Noise: A self-triggered electronics or free streaming electronics work with the threshold concept, i.e., whenever the noise in a given pad/channel exceeds the threshold, it generates a signal. In this respect, the noise signal may trigger continuously depending on the threshold value.

Every pad is connected with one of the channels of the SMX front end board, which means the electronics noise may occur at any channel. In our implementation, the noise rate has been introduced in the MuCh digitizer as a tunable parameter to perform a detailed study of the achievable data stream and to generate the realistic data stream including electronics noise. We have generated per-module noise for a period between the previous event time (t1) and the current event time (t2). The mean number of noise signals in a module is equal to the per-pad noise rate $\times$ the number of pads in the particular module $\times$ the time duration (t2 - t1). Number of noise signals in between this time duration is implemented as Poissonian distribution of mean noise. Time of each noise signal is taken as uniformly random between t1, t2 and charge of the signal is estimated based on a Gaussian distribution around the threshold charge. These generated noise signals have been added to the corresponding signal if it introduces a pileup with any earlier signal. It may be noted that the generated noise signals are not matched with any input Monte-Carlo track.

Physics Noise: Signals which are not event correlated but are produced due to physics

processes are referred to as physics noise. These are not taken care of in the event generators as these are not event correlated. One such signals are delta electrons produced by the beam particles in the target. These do not introduce a significant effect for MuCh setup, therefore, it is not implemented in the MuCh digitizer.

Thermal noise generation: Our simulation also covers thermal noise generating random signals from the readout electronics. Thermal noise is always present in the readout channel; the distribution of its value is approximately Gaussian. Whenever the noise charge exceeds the threshold, a message is generated just as in case of charge originating from a traversing particle. The occurrence of this noise is random and not correlated in time to events.

In MuCh digitizer implementation, the noise generation can be enabled or disabled and is described by a tunable parameter the noise rate per pad. From this parameter, the number of noise signals is sampled for each module for the time period from the previous to the current event time from a Poissonian distribution with the expectation value noise rate $\times$ number of pads per module $\times$ time interval. The time of the noise signals is sampled from a uniform distribution in the respective time interval; their charge is sampled from the Gaussian noise distribution. Noise signals are inserted into the readout buffer and thus into the simulated data stream. They can interfere with "real" signals from traversing particles in the same way as two "real" signals (see the previous subsection).

4.4 Investigations of time-stamped data stream

The CBM experiment intends to operate at high interaction rates of up to 10^7 events per second. At such rates, the average time between two subsequent events (100 ns for 10^7 events/s) becomes comparable to the dead time of the readout electronics (400 ns). Consequently, losses due to interference from particles originating from different events are expected. The developed simulation software as described in the previous section allows to study such

timing effects and to quantify rate-dependent losses, which is essential to determine the performance of the detector.

For the following studies, we have simulated both minimum-bias and central $Au + Au$ collisions at $p_{\text{beam}} = 10$ GeV/c. In the minimum-bias event sample, the collision impact parameter b is realistically distributed according to the geometric cross section ($\propto b\mathrm{d}b$). The time sequence of such events corresponds to the actual experimental situation. Central events ($b \sim 0$) create the highest track multiplicity and thus the highest data load on the detectors. A time series of central events only does not correspond to the physical situation, but is used here to verify the simulation software since it magnifies effects arising from the finite dead time.

The result of the MuCh simulation is a stream of digi objects integrated by the DAQ software into the streams of the other CBM detector systems. Now we can generate time stream of digis. In the framework, timeslice length and interaction rate are tunable parameters which can be provided in the `run_digi.C` macro. By default, event rate is 10^7 events per second and timeslice length is 10000 nanoseconds is set in the macro. It simulate, the free streaming scenario. Due to 10^7 Hz interaction rate, on an average events are 100 ns apart. In the `CbmRoot` system interaction is simulated with Poisson distribution. In this case approx 100 generated events will be placed in one time slice of 10000 ns length.

4.4.1 MuCh detector configuration for SIS-100 energies

The MuCh detector [7] serves for the identification of muons by filtering out other particle species in segmented absorbers. For the SIS-100 muon setup, the absorbers are arranged as five segments with three detector layers in between each pair of segments. The segmented absorber system allows to trace particle trajectories through the entire setup, which would not be possible by a monolithic absorber because of small-angle scattering of the muons in the absorber material. A thick absorber also likely to absorb muons without generating signal. Figure 4.8 shows the MuCh geometry consisting 4 detector stations and 5 abosorbers

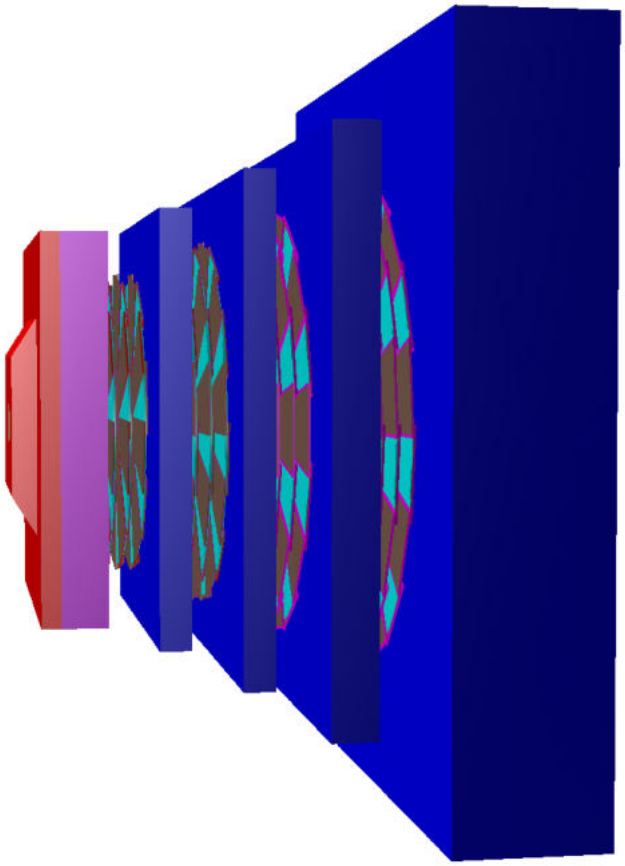

Figure 4.8: Schematic layout of the MuCh detector for SIS-100 muon setup, consisting of a set of absorber segments inter-laid with detector stations.

as implemented in current version of the `CbmRoot`. Earlier in Fig. 1.5, shown in Sec. 1.2, we showed the MuCh setup for SIS-300. Hereafter, all studies presented are based on the MuCh setup for SIS-100.

The hit rates in the detector layers differ significantly because of the successive absorption of particles by the absorber segments. Thus, different detector technologies have been employed for MuCh construction: the two upstream stations (detector triplets) will use Gas Electron Multiplier (GEM detector), while the two downstream stations will be constructed from high-rate low gain Resistive Plate Chambers (RPC). In both cases, the detector readout planes are segmented into pads in a $r - \phi$ geometry, the granularity depends on the layer position and the radial distance from the beam. The readout segmentation was optimised based on efficiency and signal-to-background ratio for the detection of muon pairs in heavy-

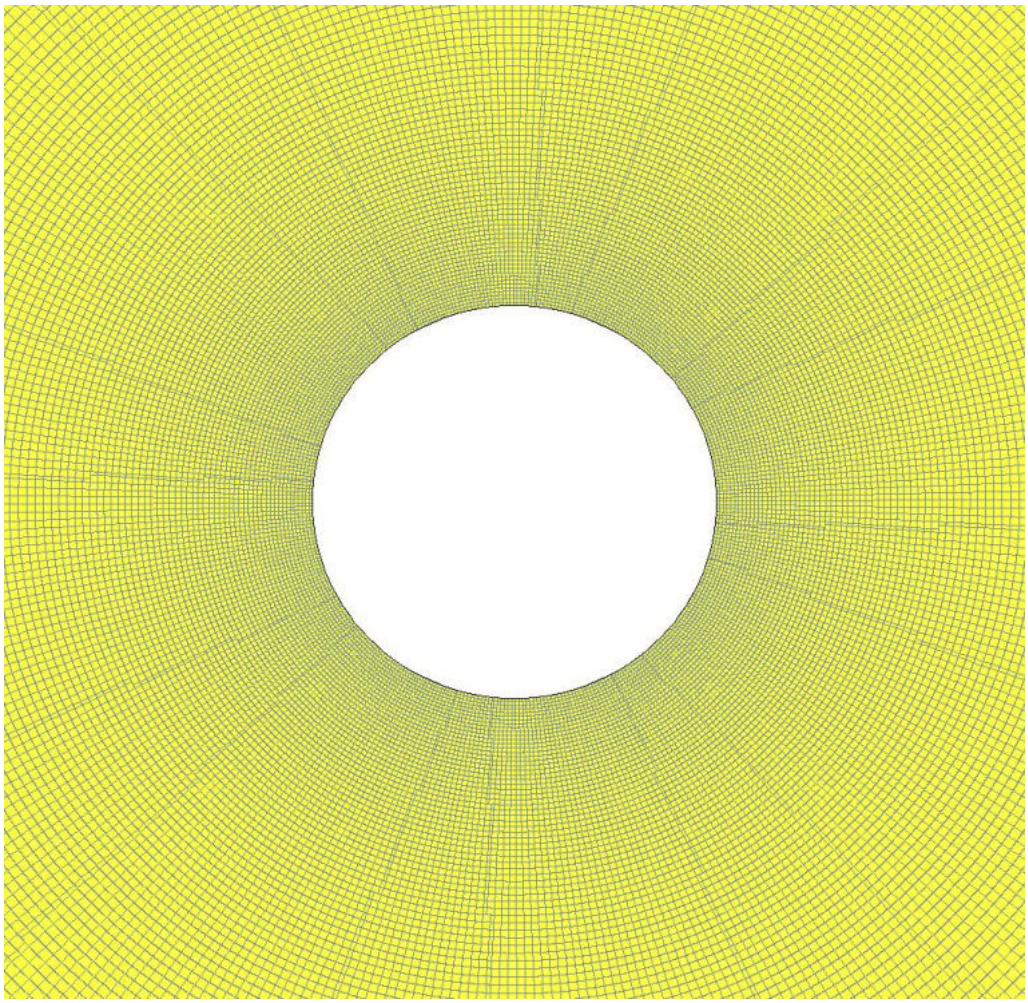

Figure 4.9: Pad segmentation of the one quarter of first layer of the first station. Here, the angular granularity is $\delta\phi = 1^\circ$ and $\delta \mathrm{r} = r\delta\phi$.

ion collisions at FAIR energies [7, 55]. The pad layout is illustrated in Fig. 4.9 as an example of the first layer of the first station. Similar pad layout but of different pad dimensions is implemented in the MuCh Geometry under `CbmRoot`. The station-1 and station-2 have 1° segmentation and station-3 and 4 have 5° and 6° segmentation respectively for all 3 detector layers under each station.

Figure 4.10 shows the digi rate obtained from the time-stamped data stream simulation combined for all 4 MuCh stations. In this data stream, there is no event boundary and all the digis have their time stamp. The MC-true event time generated by the `CbmRoot` simulation framework is overlaid for event start time representation. We see that events

can be identified by peaks in the distribution of digis [56]. The event building has been discussed as outlook in the conclusion Sec. 5.1.

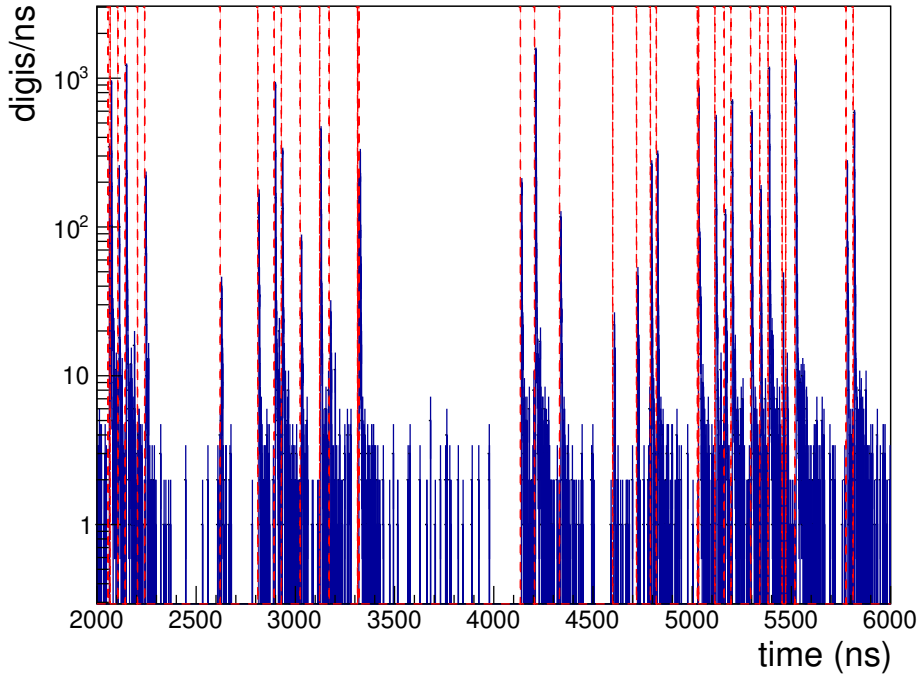

Figure 4.10: The time distribution of the MuCh digis for a certain time period (within a time slice), with several events in and with moderate noise switched on, resembling self-triggered free running data stream. Overlaid MC true event start time as red dashed line.

4.4.2 Pile-up effect

The rate-dependent data losses can also be qualified by studying the pile-up effects. In our simulation, as earlier mentioned, the reference to the Monte-Carlo origin is kept for each digi object. Pile-up thus happened if a digi object has references to more than one Monte-Carlo particles. We define the pile-up fraction as the number of digis with multiple MC references divided by the total number of digis. As discussed in Sec. 4.3, SMX chip is continuously being upgraded and the efforts are going on in the direction of reducing the dead-time, however effective dead-time of SMX [54] for MuCh mode will be upto 400 nano-second only and therefore all the studies presented in this thesis have been performed using

400 nano-second of deadtime.

A common detector performance figure is the occupancy, conventionally defined as the probability for a single channel to be activated in one event. In our free-running scenario, we calculate this quantity as the number of digis divided by the number of pads and the number of simulated events.

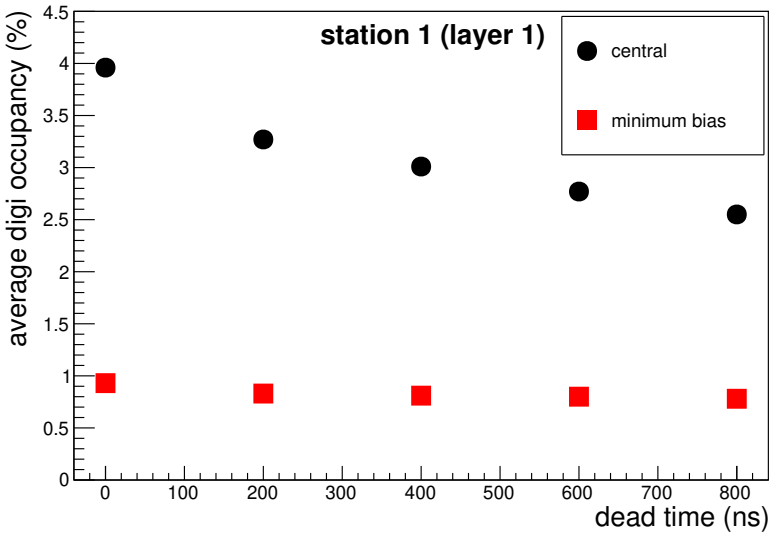

Figure 4.11: Average single-channel occupancy in the first layer of the first station as a function of dead time for Au+Au collisions at 10^7 events/s.

Figure 4.11 shows this occupancy (averaged over one detector layer) for the first layer of the first station at an event rate of 10^7/s as a function of the single-channel dead time. The decrease in occupancy with dead time, moderate for minimum-bias events and better visible for central events, is an expected consequence of the data loss due to signals arriving in the dead time of a previous signal.

Pile-up fraction is shown in Fig. 4.12 again as a function of dead time for an interaction rate of 10^7/s. For minimum-bias events, the fraction slowly varies with dead time; for a value of 400 ns, corresponding to the current SMX design, it amounts to about 13%. It should be mentioned that as discussed later the maximum contribution to this pile-up fraction is from the secondary tracks generated in the material. This does not represent the pile-up for all

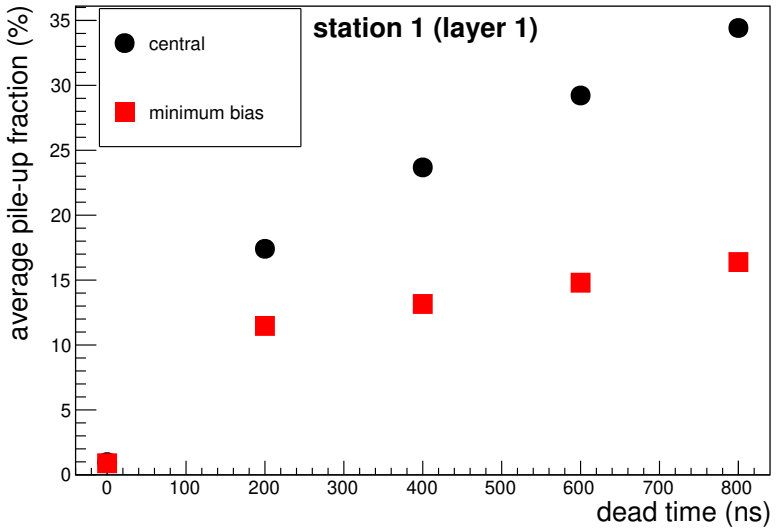

Figure 4.12: Average pile-up fraction in the first layer of the first station as a function of dead time for Au+Au collisions at 10^7 events/s.

tracks (primary & secondary) associated with the incident primaries, which will govern the capabilities of tracking of primary tracks in the detector.

The pile-up fraction is not dependent on centrality therefore change in pile-up fraction from central to minimum bias is not directly correlated as for occupancy. Similarly from event mode to time stream mode, the pile up effect increases but not consistent for different stations. As an outlook, we will investigate in detail about this behaviour corresponding to physics observable.

The pile-up fraction is shown in Fig. 4.13 differentially for each sector in the first layer of the first station for different interaction rates, varying from 10^4/s to 10^7/s. Here one sector denotes one ring containing 360 pads which is according to progressive geometry (see Fig. 4.9). Again, the rate effect is better visible for central events (right-hand panel) than for minimum-bias events (left-hand panel). Pile-up at low rates happens between tracks within the same event and is thus irreducible by varying the dead time; it corresponds to conventional "double hits" and is fixed by the granularity (pad layout) of the detector. From an event rate of about 10^6/s on, the additional pile-up between tracks from different events

becomes visible and increases with increasing rate.

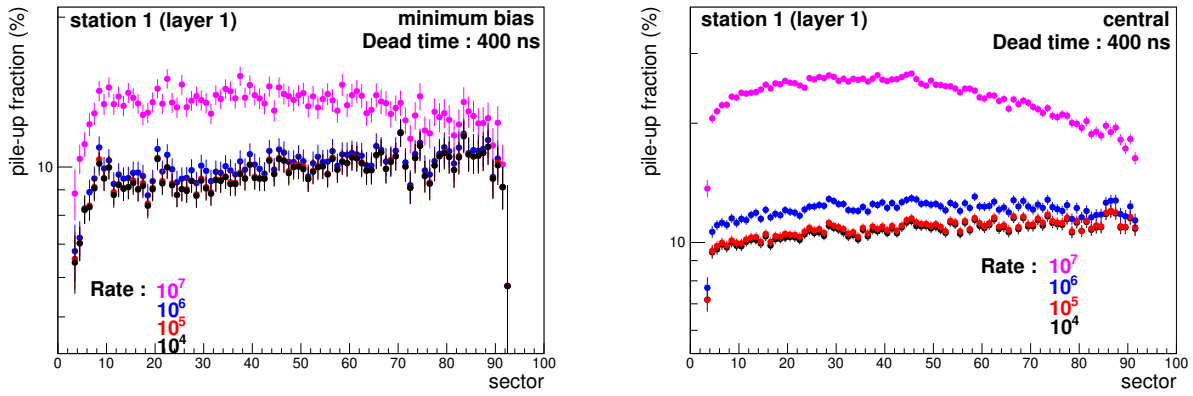

Figure 4.13: Pile-up fraction in each sector of the first layer of the fist MuCh station for a dead time of 400 ns and for different interaction rates. The left panel shows the (realistic) case of minimum-bias events, the right panel the (artificial) case of central events only.

Both occupancy and pile-up studies have been performed using central events to see the extreme effect along with that for the minimum bias events. Fig. 4.14 shows the occupancy distribution and Fig. 4.15 shows the timing inefficiency or pile-up effect as a function of the sector number representing the radial ring as per readout segmentation for event-by-event and time-based simulation for the first layers of all 4 MuCh stations. The pile up plots in Fig. 4.15 show that for central events, pile ups on the first station accounts up to $\sim$ 26% for time stream mode compared to $\sim$ 10% in the event-by-event mode. Though, the effect is lesser in minimum bias events which is realistic, however, it is significant.

In Sec. 4.2, it is mentioned that the segmentation parameters of RPC based 3^{rd} and 4^{th} MuCh stations are not similar to that of the first two stations, therefore the occupancy and pile up effect is not similar for the third and the fourth stations as compared to the first two stations.

The Occupancy and the timing inefficiency plots with varying dead time have been studied in detail. It is found that the effect of varying dead-time is not visible in the event by event mode as most of the particle in an event arrives at the same time therefore dead time does not play any significant role. However, for the time-based mode, increasing dead-

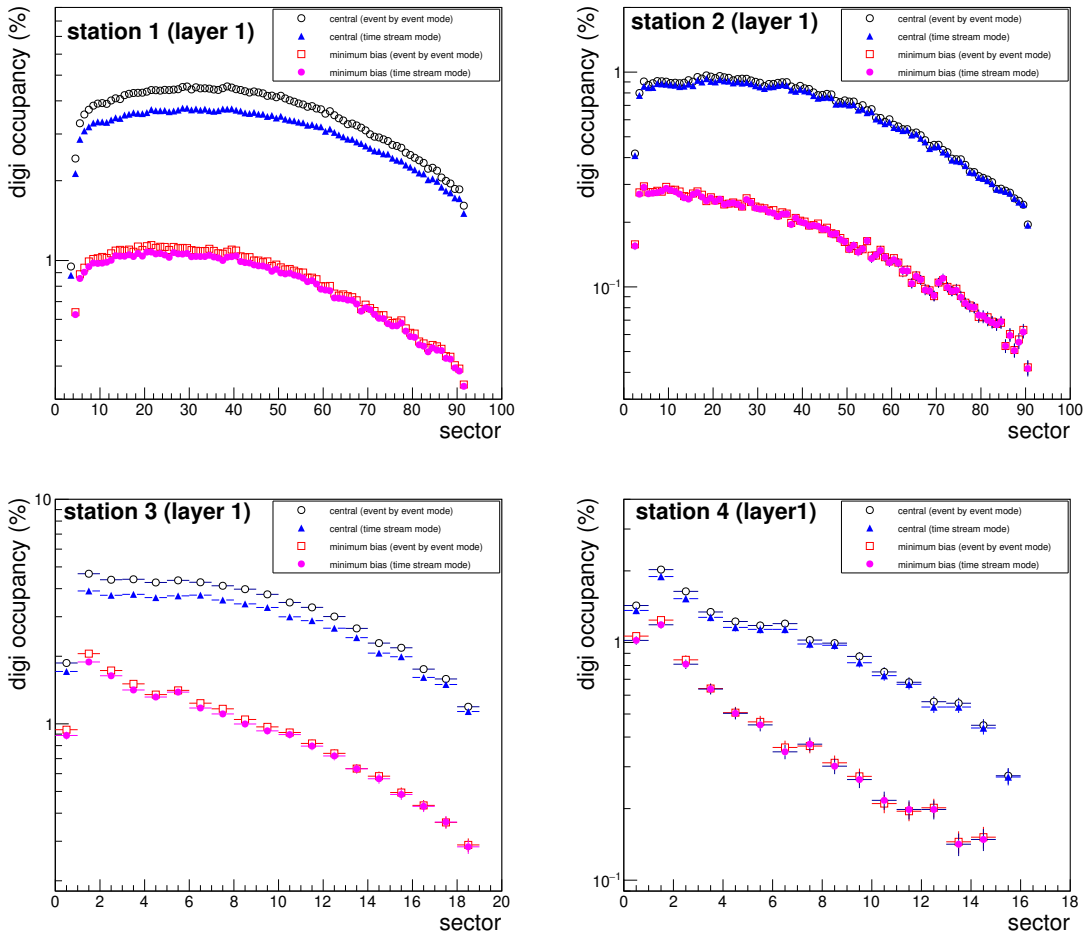

Figure 4.14: Occupancy with respect to sector for the 1st layers of all 4 MuCh Stations. Occupancy is compared for the time based mode and the event mode for both central and min-bias events.

time decreases the digi occupancy as once pad is fired it is active for minimum upto next 3 events and could not be fired again if track passes to the same detector channel therefore not counted for the occupancy for the later cases.

The situation in the other three MuCh stations is comparable to the one in the first station, as shown in Fig. 4.16. In all stations, an increase of inter-event pile-up with rate is observed on top of the irreducible in-event pile-up. Note that for constructional reasons, station-2 has the same angular pad segmentation as station-1, although the hit density is significantly reduced in the absorber segment in between which can be seen in the left panel

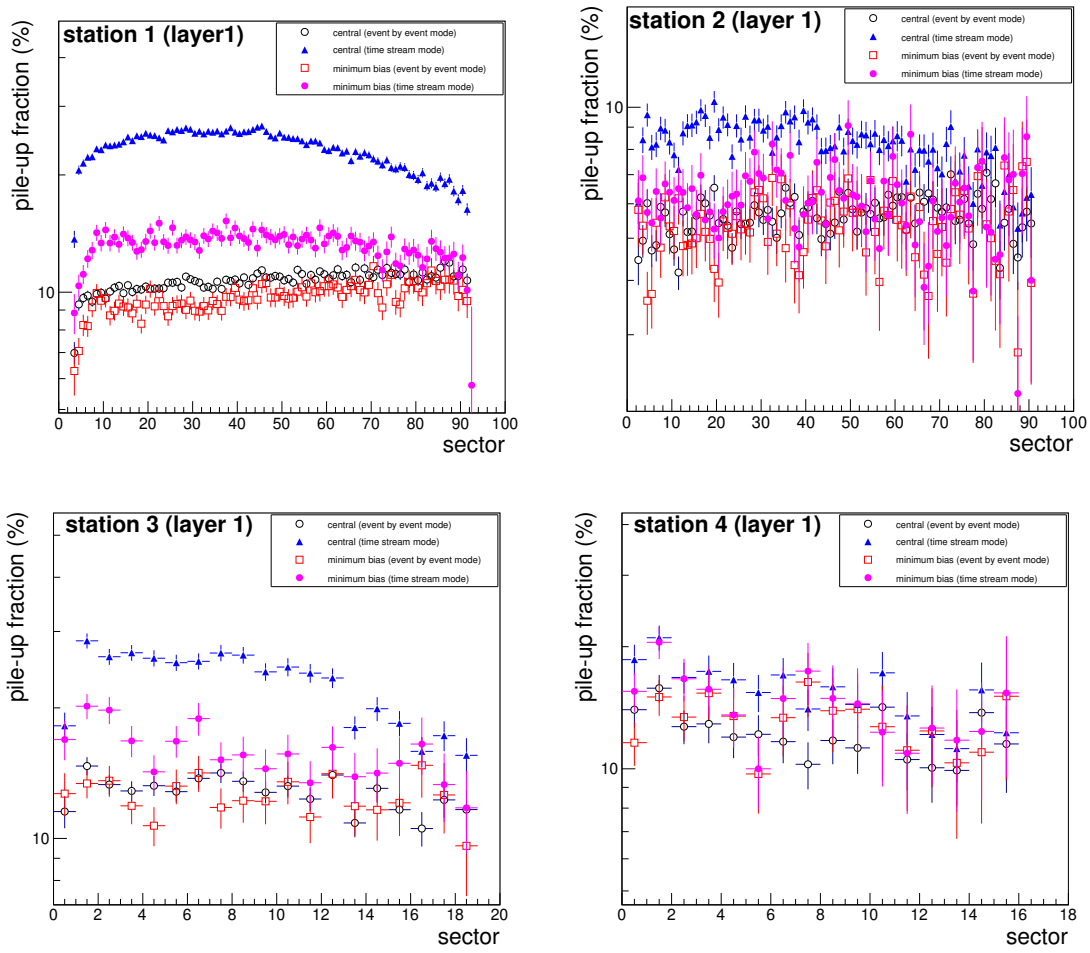

Figure 4.15: Pile-up fraction for event by event and time stream mode for 4 stations of MuCh setup. Dead time is 400 ns and event rate is 10 MHz.

of Fig. 4.17. This results in a lower occupancy and a lower amount of pile-up losses compared to station-1. The pad layouts of stations-3 and 4 were chosen so as to result in an occupancy similar that to in station-1. The right panel of Fig. 4.17 shows the digi occupancy for first layer of 3^{rd} and 4^{th} MuCh station as a function of the sector number.

The consequence of the pile-up for the sensitivity of the detector to physics observables must be assessed by proper simulations including the full reconstruction of the simulated data stream in terms of hits and tracks. Such studies are yet to come. However, identifying the Monte-Carlo origin of the pile-up cases, we find that 92.47% of all pile-up cases happen between two secondary tracks in a shower created close to the downstream edge of an

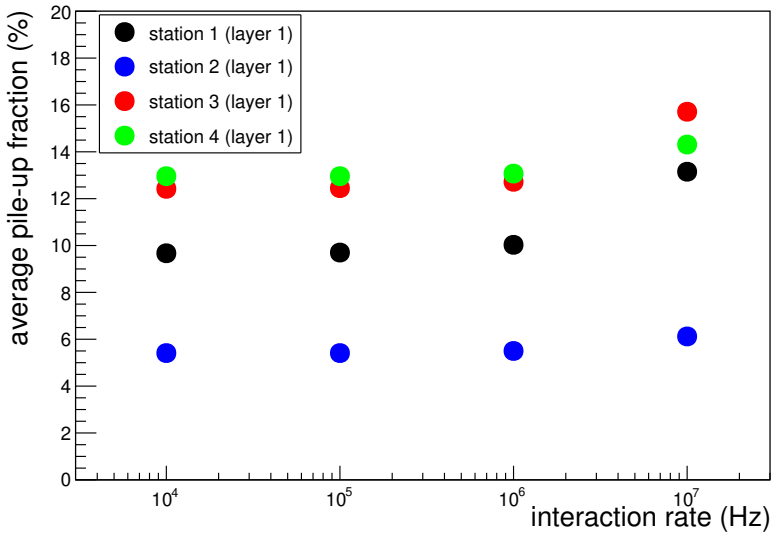

Figure 4.16: Average pile-up fraction as a function of interaction rate for minimum-bias Au+Au collisions at 10AGeV/c. The single-channel dead time was taken to be 400 ns.

absorber layer. Such shower tracks concentrated in a small emission cone are thus more likely to deliver charge into the same read-out pad. About 7.4% of pile-ups occur between a primary and a secondary track; the fraction of pile-ups between two primary tracks (from the main event vertex) is negligible. This suggests the impact of pile-up on the physics performance of the detector to be minimal.

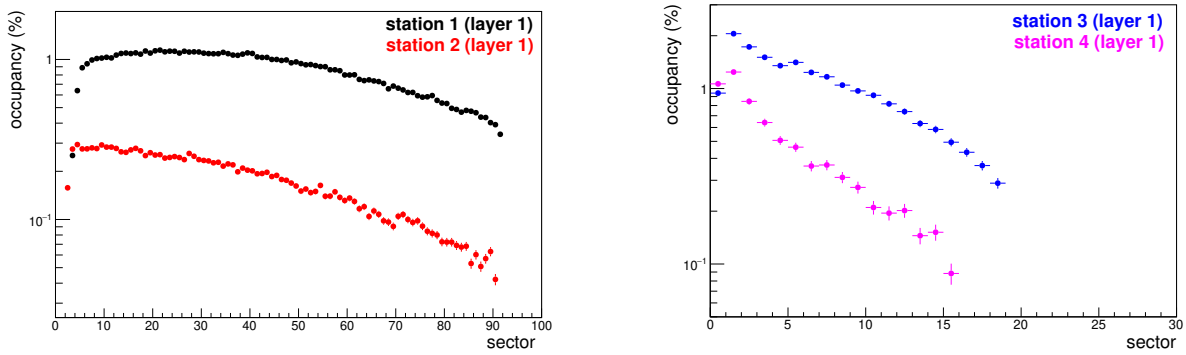

Figure 4.17: Digi occupancy as a function of the sector number of the first layer of 1^{st} and 2^{nd} MuCh stations (left panel) and 3^{rd} and 4^{th} MuCh stations (right panel).

4.5 Cluster and hit finding

In reconstruction, the cluster and the hit finding process, is the first reconstruction task, to be executed before track finding as shown in Fig. 4.3. There are two probable approaches for event building from the time-stream data (a) performing event building based on the digi multiplicities, (b) event building using tracking [57, 56]. Both approaches need precise cluster and hit finding procedure and should thus be independent of the size of the time stream delivered by the data acquisition and should not depend on the event by event processing. A precise process has been developed to find clusters and hits with respect to time measurement for the MuCh setup. First, using both spatial and temporal information, neighbouring fired pads are grouped to form a cluster. The centroid of a cluster is then used to construct a hit. The implementation sorts the input raw data (digis) into a `std::vector` for each module. Each detector layer under every MuCh station is divided into a few modules. The module size is according to the availability of the largest width GEM foil and production feasibility, thus, number of modules in a layer varies for each station. Cluster and hit findings are independent of the module, thus can be executed in parallel for each module [58]. The corresponding classes are developed in such a way that it can be used for both the event-by-event and the time-based mode and compliant to free-streaming data.

The DAQ will deliver time sorted data stream, therefore simulation framework as described in Sec. 4.2 also stores all the digis in time sorted manner, which will be input for the cluster and hit reconstruction tasks. At the beginning, the cluster finding process intermediately stores module wise digis information into a vector, and thereafter, looped over all digis of a particular module. Different time windows, based on the tunable parameter of cluster separation time, have been built for further processing. It helps to maintain uniform execution time performance for the event by event and time stream mode. Thereafter, it is looped over generated time windows and enable fired pads (channels) in that vector. After this, it is looped over on the fired pad vector to create clusters accordingly. Keeping online reconstruction in mind, the entire cluster and hit finding processes have been developed

based on a modular design.

Clusters generated from the cluster finding algorithm are used to generate hits (real space points). Detailed performance of hit finding algorithm has been performed using three types of hit finding approaches (a) one hit per pad (b) one hit per cluster and (c) search for local maxima. The search for local maxima has been seen as the most realistic hit finding approach and used as default algorithm in the MuCh simulations [7]. Hereafter, cluster-by-cluster search for local maxima, in terms of the charge distribution stored as ADC value, has been performed. Hit coordinates are assigned to the centre of gravity based on the ADC values of fired pads of a cluster, corresponding to local maxima. If cluster dimensions are less than equal to 2 pads then one hit is created. The spatial position is determined based on charge distribution on the pads of a particular cluster which is based on eq 4.1 and eq 4.2. For time-based reconstruction, timestamp of the hit is important and hit time is the minimum time of the digi out of all the digis under the particular cluster as per eq 4.3.

$$X_{hit} = \frac{\sum_i (x_i \times q_i)}{\sum_i (q_i)} \tag{4.1}$$

$$Y_{hit} = \frac{\sum_i (y_i \times q_i)}{\sum_i (q_i)} \tag{4.2}$$

$$T = \min(t_i) \tag{4.3}$$

All the hardware messages (hits information) given by the data acquisition system (DAQ) [59] are gathered in time stream in the form of data packages (known as time slices). Time slice contains data for a pre-defined fixed time interval, which will comprise many events (O(1000) - O(100000)). All reconstruction algorithms have to operate on such time slices as is the raw data input format. In the context of the processing time of such time streams, its execution time should thus be independent of the size of the time slice

delivered by the data acquisition, making it suitable for being used in the online reconstruction. Above mentioned cluster and hit reconstruction processes have been optimized keeping this requirement in consideration. The Figure 4.18 shows the per event execution time as a function of time slice length (number of events per time slice) and it shows per event cluster and hit reconstruction execution time is independent of the number of events under a time slice.

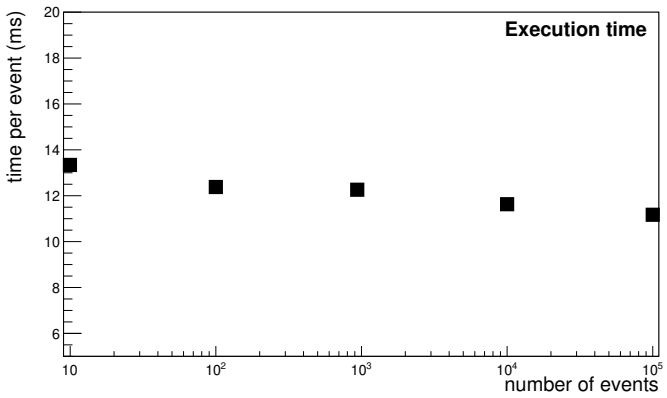

Figure 4.18: Execution time for cluster and hit finding.

In the latest version of `CbmRoot`, few example macros are available which guide to generate time-stampled data stream. One such common example macro is *run_ digi.C*, in which the output time-slice length and interaction rate can be set as input parameters. To further visualise our time based simulation study, data stream equivalent to real experiment-like scenario has been generated with very low interaction rate e.g. 10^5 and time slice length of 10000 ns. In this case, most of the generated events should be placed such that one event fall in one time slice.

Figure 4.19 shows the distribution of the number of MuCh digis for simulated time-slices generated by providing 1000 background events as input. According to the selected input combination of interaction rate and time-slice length, one event should fall under one time-slice, but due to Poissonian distribution of event start time, few time-slices are containing two or three events and similarly few time-slices are containing none of the event.

The time distribution of digis within a single event is shown in Fig. 4.20 together with

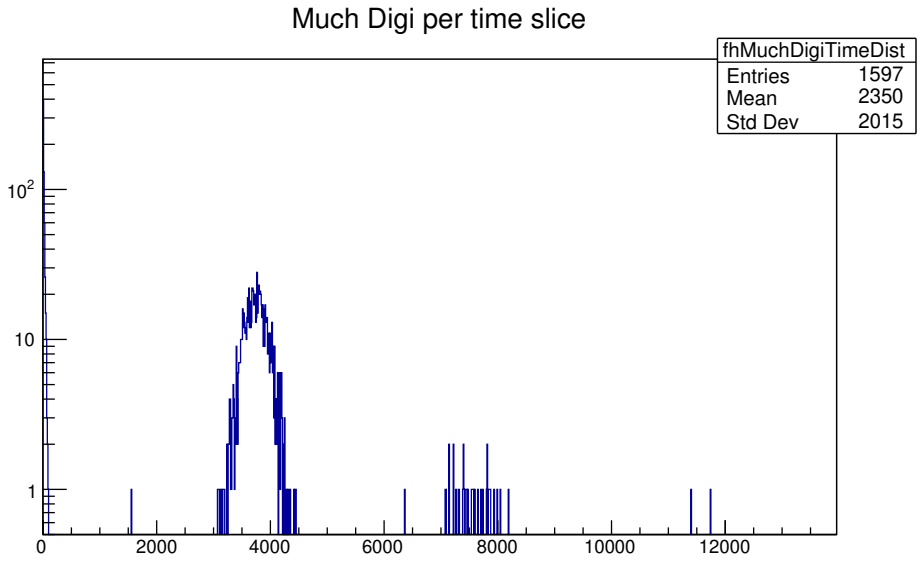

Figure 4.19: The MuCh digi distribution per time slice for simulated 1000 events. With generation of time-slices, correspondence to event is broken.

the corresponding Monte-Carlo true time of the track in the detector. Included is also the reconstructed hit time, where a hit results from a cluster-finding procedure which groups simultaneously active neighbouring pads. The MC true time shows a narrow peak and an approximately exponential tail. The peak originates from both primary and secondary tracks, which are registered within a time span of less than 10 ns. Due to interaction with absorber material of the muon system, 94.5% digis, registered in MuCh detector layers, are generated from secondary tracks. The offset of the peak (about 7 ns) reflects the time-of-flight from the main interaction vertex to the MuCh detectors (z distance of first MuCh detector layer is shown in Fig. 1.5). The exponential tail comprises secondary tracks only. In comparison, the digi time distribution shows a broadening of the peak due to the time resolution and an additional offset stemming from the drift time of the electrons in the drift gap of the GEM counters (see Fig. 4.5). Note that the integrals of the distributions of MC points and digis are different since one track can activate more than one pad and more than one MC points can generates one digi. The hit time, on the other hand, is corrected for the electron drift and thus shows no offset with respect to the MC origin.

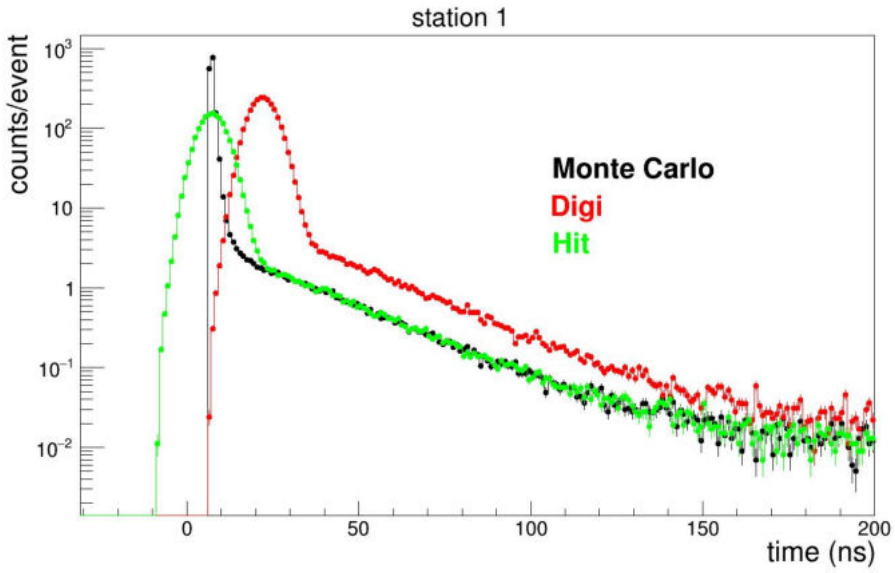

Figure 4.20: The time distribution of MuCh digis in a single event.

4.6 Hit Reconstruction Validation

Correctness or validation of the generated hit time after reconstruction including digitization, clusterization and hit creation needs to be evaluated with respect to time after transportation. Monte Carlo time is the time of flight. Each MC-Point contains time in nanosecond which is the time elapsed since the beginning of the event till the start of the point. For event by event processing, interaction time = 0 for each event. Digi time is computed as the sum of event time, MC-Point time and drift time (due to the underlying detector as described in detail in the Sec. 4.2). As explained in Sec. 4.5 hits are created after clustering of digis. During the computation of hit time, correction of drift time is taken care such that the global hit time is independent of the underlying detector effect.

The Figure 4.21 shows the time measurement for all 4 MuCh stations after 3 different stages (i) transport simulation (black), (ii) digitization (red) and (iii) hit reconstruction (green). It shows that almost all the particles reach a particular station at the same MC time (denoted with black colour) and varies according to time of flight to the particular station. It can be seen that time after digitization (red colour) shifts towards right due to

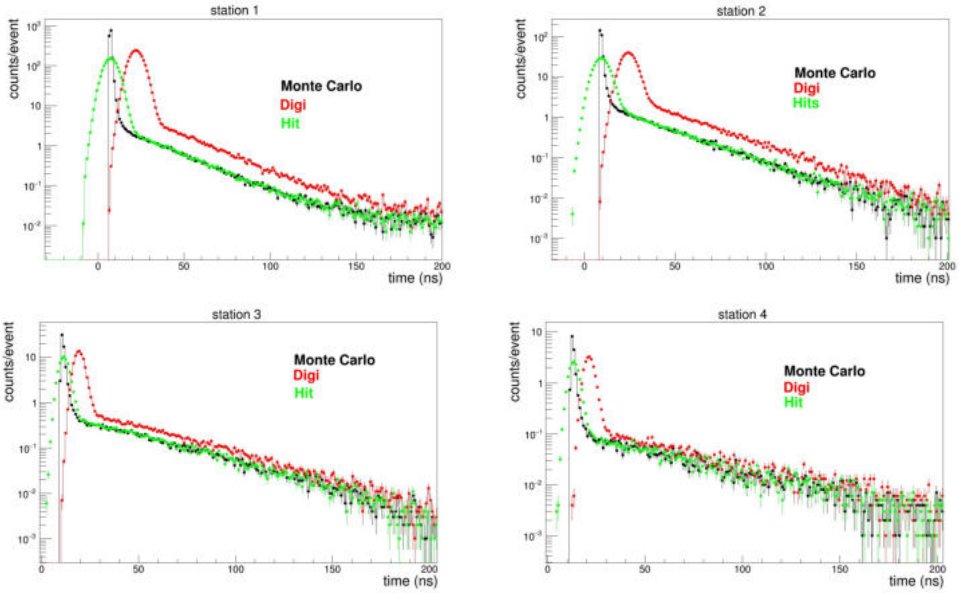

Figure 4.21: Time distribution of Monte-Carlo points (black), digis (red) and hits (green) in a single event in the first layer of all 4 MuCh stations of the MuCh detector (top left station 1, top right station 2, bottom left station 3, bottom right station 4). Hits are reconstructed from clusters of digis in neighbouring active pads. The origin of the time axis corresponds to the Monte-Carlo event time.

the drift time of the underlying detector. The peak of the hit time (green colour) coincide with the MC time as the drift time correction performed during hit reconstruction. The long tail of time distribution is due to slow momentum and late particles.

To validate the time measurement process, masked the introduction of drift time in the digitization and also removed the correction done in the hit reconstruction. All the plots of Fig. 4.21 have been regenerated for all 4 MuCh stations to qualify the correctness of the process.

The Figure 4.22 shows time measurement after masking of drift time effect for all 4 MuCh stations. It can be seen that MC time, digi time and hit time coincide with each other. Hits multiplicity is slightly lesser than MC point multiplicity due to hit finding algorithm which is based on the search for local maxima, not the one-hit per fired pad.

Validation of simulated time measurement can be evaluated via residual and pull distribution with respect to time. Residual time is defined as the reconstructed time (hit time)

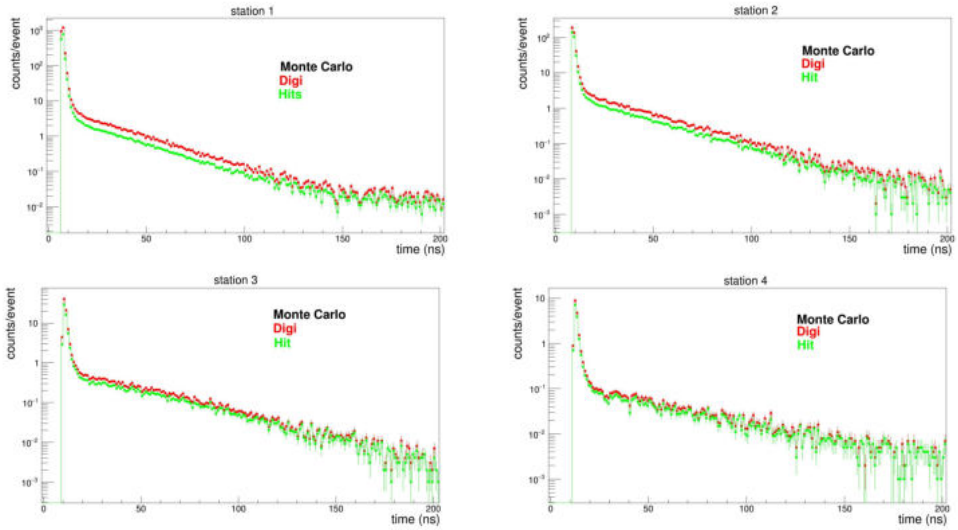

Figure 4.22: Time measurement for all 4 MuCh stations after transportation (monte carlo time (black)), after digitizatoin (digi time(red)) and after hit reconstruction (hit time(green)) for (a) station 1, (b) station 2, (c) station 3, (d) station 4.

minus MC time generated by transport simulation and pull distribution is generated after the division of individual time error associated with each hit on the residual time. The Figure 4.23 shows residual and pull distribution which is Gaussian in shape due to detector drift time effect (black), however, without drift time distribution is peaked at 0 (blue) which shows that reconstruction is flawless with respect to time measurement.

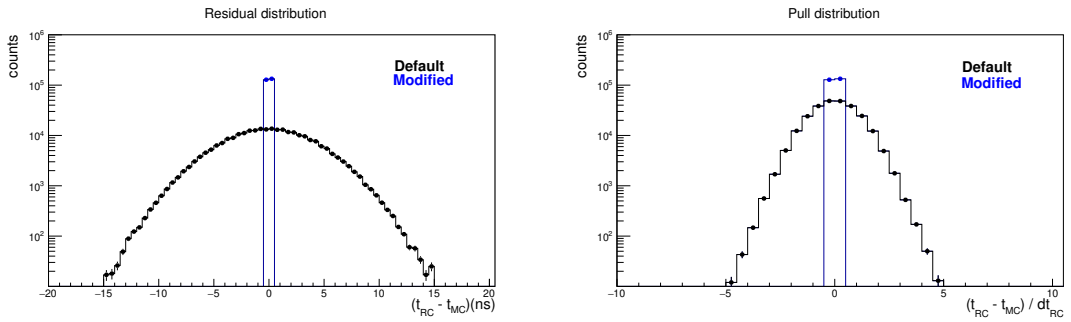

Figure 4.23: Residual (left) and pull (right) distribution for the entire MuCh setup.

Chapter 5

Summary and Discussions

The computing demands for modern nuclear physics experiments are increasing in manyfold, this holds true in particular for experiments relying heavily on real-time data processing. The Compressed Baryonic Matter experiment will probe strongly interacting matter at extreme net-baryon densities. Its ambitious physics programme requires to take data in free-streaming mode at an unprecedented interaction rate of 10 MHz, thus, event building, on-line event selection, trigger generation among others will be performed on the computers at real time only. One can say that the CBM is in one hand a mega physics experiment and a big computing project on the other.

In CBM, during proton or heavy ion induced collisions, J/ψ mesons will be generated with an extremely low production cross section. As J/ψ decays dominantly into the dimuon channel of $\mu+$ and $\mu-$, a Muon Chamber (MuCh) system is being designed and developed for the detection of dimuon pairs originating from these collisions. It consists of an alternating layers of segmented absorbers and detector stations. As collision rate is extremely high and the rate of J/ψ production is very low at FAIR energies ($E_L = 10 - 40$ AGeV), it has motivated us, to develop a real-time event selection process. The process selects events which are likely to contain J/ψ. In this thesis we have described following in detail,

1. Development of an event selection algorithm also known as trigger algorithm for the

CBM-MuCh detector data which decides based only on the hits of the last MuCh station whether the specific event likely to contain J/ψ or not. This goal leads to the development of two event selection algorithms, (a) Brute-Force and (b) Selective based on the di-muon signature.

2. A systematic study for the implementation of the event selection process using different parallel computing paradigms using heterogeneous computing has been performed and it has been demonstrated that the events containing J/ψ can be selected on-line using single machine comprising GPU and CPU.

3. Development of a time based software stack to perform trigger-less real time collision simulation and generated realistic data stream.

To select candidate events in real-time which contain J/ψ decaying into $\mu^+\mu^-$, the Brute-Force and the Selective event selection algorithms have been developed on the basis of the signature that the two daughter muons of high momenta traverse all the absorber layers of MuCh and reach the trigger station, while hadrons, electrons, and low-momentum muons will be absorbed before reaching the last station. Since J/ψ decays promptly ($c\tau = 7.1 \cdot 10^{-21}$s), the decay products practically originate from the primary (collision) vertex, i.e., from the target. Owing again to the high momenta of the muons, their trajectories can be approximated by straight lines even in the bending plane of the dipole magnetic field, which has a bending power of 1 Tm. The trigger station consisting of three detector layers provides three space points for a muon, allowing to check the back-pointing to the primary vertex. The signature of a candidate event is thus the simultaneous registration of two particles in the trigger station which can be extrapolated backward to the target. The Brute-Force algorithm processes all combinations of space points of the last 3 layers of MuCh and the Selective algorithm processes only selective combinations which follow tolerance range criteria, thus algorithmic complexity reduces by an order for the Selective algorithm than the Brute-Force algorithm.

It has also been demonstrated that using different parallel computing paradigms like pthread, OpenMP, MPI, and OpenCL for multi-core CPU architectures, and CUDA and OpenCL for many-core architectures like NVIDIA GPUs, processing at an interaction rate of 10^7 events per second can be achieved by the studied event selection algorithm. For both platforms, the event selection procedure suppresses the archival data rate by almost two orders of magnitude without reducing the signal efficiency, thus satisfying the CBM requirements for high-rate data taking and demonstrated that events containing J/ψ can be selected on-line using available GPU and CPU.

On GPUs, we have found a speed-up of 4.5 with respect to the optimised single-thread execution on CPU. This result, however, is only obtained after careful optimisation of the implementation in CUDA. OpenCL on NVIDIA GPUs are found to perform slightly worse than that for CUDA. The execution time measurements for the Brute-Force event selection process show that about $3 \cdot 10^5$ events per second can be processed on a single GPU card of NVIDIA Tesla family. It is shown that the compute complexity of the Brute-Force algorithm is O(n^3), and O(n^2) for the Selective algorithm. The complexity of the selective algorithm has been reduced by an order of magnitude and therefore shows that more than 10^7 events per second can be processed on a single GPU card and also using a single machine with dual CPU. Present hardware supports up to eight GPUs on a single motherboard. This suggests that other real-time processing like event building may also be accommodated together with our event selection for the targeted CBM interaction rate of 10^7 events per second.

In a multi-core CPU environment, we have compared OpenCL, pthread, OpenMP and MPI as open-source concurrency paradigms. A linear scaling of the data throughput with the number of parallel threads is observed up to the number of available physical cores. For the Brute-Force event selection process, in the powerful S2 setup consisting of the total 64 AMD cores, we find that about $2 \cdot 10^6$ events can be processed per second, which is close to the targeted event rate of 10^7/s. This demonstrates that the SIMD instructions provided by the modern CPUs are essential to achieve the required throughput. It is also

shown that the computing demands of the CBM experiment for the real-time selection of J/ψ candidate events can be achieved by properly making use of the parallel capacities of the heterogeneous computing architectures. As an example, the NVIDIA Tesla GPU of the S1 setup could be placed into the S2 setup to achieve the desired goal.

By comparing the different programming paradigms, we find that the cross-platform OpenCL to be a proper choice for the heterogeneous computing environments typical for modern architectures, which combine CPU cores with GPU-like accelerator cards. For such kind of systems, OpenCL provides a suitable solution to simultaneously exploit all available compute units for a given application. It also provides the flexibility to future improvements in computing architectures, which is of particular importance for CBM as an experiment in the construction stage. This flexibility, however, comes at the price of a reduced performance on CPU when compared to pure parallel programming paradigms.

After detailed investigations of heterogeneous computing and available parallel computing paradigms, we have also presented a time-based simulation scheme for the MuCh detector of the CBM experiment which provides a realistic detector response of detector and read-out electronics in the environment of a self-riggered, free running data acquisition system. The key to this scheme is to enable interference of tracks from different events with small time separation, using an intelligent buffering procedure and a proper prescription for the treatment of signals arriving close in time in the same read-out channel (pad). The described MuCh simulation software is integrated into the common cbmroot simulation framework, which produces a data stream similar to that expected from the real experiment. The treatment of thermal, uncorrelated noise has also been incorporated.

The simulation software allows us to study the performance of the CBM-MuCh detector in a realistic scenario of a time sequence of minimum-bias nuclear collisions. In particular, rate-dependent effects like data losses arising from pile-up between subsequent events can be addressed, which is essential to influence the reconstruction efficiency of the detector under various operating conditions. Our findings of pile-ups as a function of the electronics dead

time and of the interaction rate are in qualitative agreement with our expectations, which to a certain extent verifies the software implementation. Quantitatively, we see significant losses from inter-event pile-up, over and above the irreducible in-event double hits, for interaction rates from 10^6/s onwards.

The developed software also allows to optimize the read-out of the detector in terms of parameters like single-channel dead time, time resolution or the applied threshold, thus providing feedback to the designers of the read-out SMX ASIC. The software will be validated against the real detector behaviour using data from the laboratory and the in-beam tests of detector prototypes. In particular, we plan to improve the currently simplified treatment of double hits in an ASIC channel, replacing the now used theta function with the real signal shape in the slow channel of the SMX.

The quantitative effects of the rate-dependent data losses on the sensitivity of the detector to physics observables will be the subject of further studies. This requires the application of the full, time-based reconstruction (hit and track finding) on the generated data stream. The simulation software described in the thesis provides the required tools for such studies.

In HEP experiments, cluster and hit finding is the first step in the reconstruction. In the trigger-less experiments like CBM, output from the experiment is time-slice, thus processing time of cluster and hit finding step should not vary with the size of event or time slice. It has been demonstrated that the computational performance of the hit finder meets this design requirement and per event execution time is constant. Further speed-up of the cluster and hit finding algorithms will be possible via data-level parallelisation. In a nut shell, we have described the development of the entire time-based simulation approach for the CBM-MuCh detector. It is shown that the framework is ready with realistic time based measurements. In the CBM collaboration, simulation studies being performed are based on time stream mode and detailed comparisons are being performed between the event-by-event and the time-stream mode.

5.1 Outlook

The event selection procedure developed and investigated in the first part of thesis relies on data aggregated into events, corresponding to a single nucleus-nucleus interaction. The data acquisition of the CBM experiment, however, will deliver free-streaming data not associated to a single event by hardware trigger. To properly account for this situation, not only the spatial coordinates, but also the time measurement of each hit must be considered for the event selection. This will increase the complexity of the current, rather simple algorithm.

We are working together with the CBM collaboration towards extending the algorithm to event building and selection from the real online data stream and also will investigate the throughput on multi-core and many-core platforms in parallel using hybrid programming [60]. In addition, other algorithmic approaches to the trigger problem will be investigated, reducing the combinatorics by a more selective triplet construction.

Our study shows that the computational problem can be solved with a reasonable expenditure on CPUs, but also on GPUs as co-processors, or by a combination of both [61, 62]. It does not yet include a full exploitation of possible measures for further acceleration, like using vendor-specific compilers (Intel) or using manual code vectorization. Such investigations will be performed in the future as prerequisites for a decision on the hardware architecture, which of course will have to have balanced performance with acquisition and running costs.

After simulating the free-streaming data as expected to be delivered from the DAQ system of the running experiment, the event building process is to be implemented in the current framework as the entire reconstruction process is fully oriented on an event-by-event process.

There are multiple approaches for the event building [56]. For example, an approach could be based on dip detection on the free data stream [57]. In this approach, the event building process can be performed with respect to MuCh and thereafter the same can be adopted for the other detector systems. Once the event boundary is defined, thereafter, all reconstruction algorithms, working on an event-by-event base, can be used without any

modification. A very preliminary event building flowchart from time-slice data stream is shown here in Fig. 5.1.

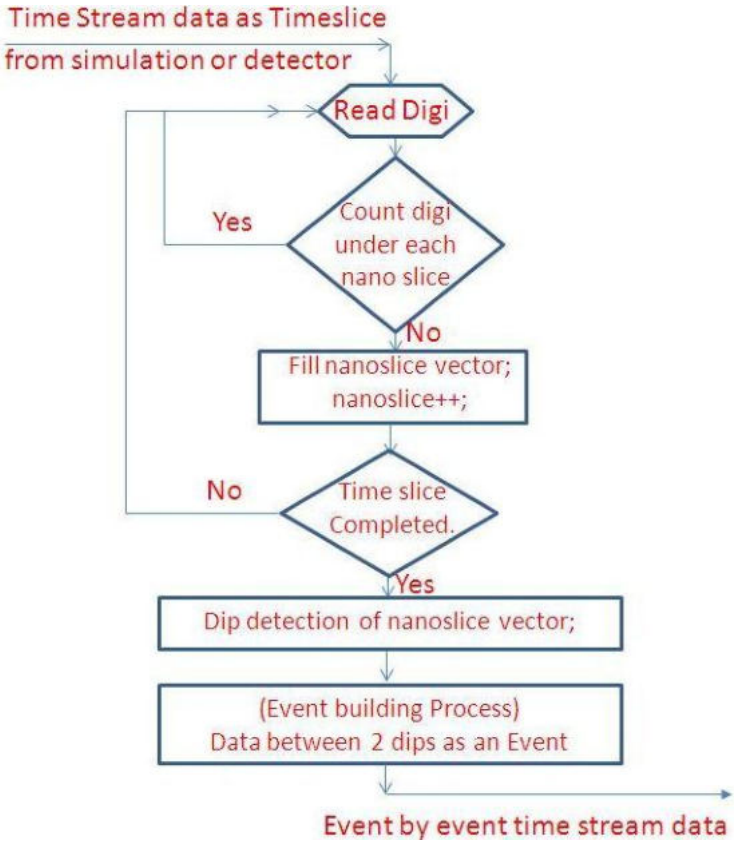

Figure 5.1: Proposed event building flowchart

We are working together with the CBM collaboration towards multi-level event building approaches from the real online data stream such that events could be generated precisely. Due to the extreme event rate, huge amount of data are expected to be produced. To use multi-core processing and online reconstruction of a continuous data stream, it requires modifications to the framework and also to the data model which are currently based on `TClonesArrays` of ROOT framework. Discussions and modifications are going on towards using `std::vector` as data containers instead of `TClonesArray`.

5.2 Achievements

After a brief summary and outlook, we enlist here major achievements in the thesis,

1. Development and implementation of a first-level event selection (FLES) process for CBM-MuCh using the Brute-Force and the Selective algorithms. Performed a comparison of performance between the two algorithms.

2. The developed event selection procedure suppresses the archival data rate by almost two orders of magnitude without reducing the signal efficiency, thus satisfying the CBM requirements for high-rate data taking [12].

3. A speed-up of 4.5 for event selection process using NVidia GPU is achieved in comparison with execution time of the optimized single-thread execution on CPU. (The results are published in Computer Physics Communications journal.)

4. Results show that using the Brute Force Algorithm, about $3 \cdot 10^5$ events per second and using the Selective Algorithm about 10^7 events per second can be processed on a single GPU card of NVIDIA Tesla family. (The study has been presented in an international conference.)

5. Systematic study of heterogeneous architecture and different parallel computing paradigms have been presented and performance comparison of the algorithms on CPU, GPU using pthread, OpenMP, MPI, CUDA and OpenCL has been performed. It is found that the cross-platform OpenCL is a proper choice for heterogeneous computing environments typical for modern architectures, which combines CPU cores with GPU-like accelerator cards.

6. By the throughput obtained for the selection of J/ψ candidate event, it is determined that the computing demands of the CBM experiment can be achieved by properly making use of the parallel heterogeneous computing architectures.

7. Development of a time-based signal generation scheme for the Muon Chamber simulation by enabling interference of tracks from different events with small time separation, using an intelligent buffering procedure and a proper prescription for the treatment of signals arriving close in time in the same read-out channel (pad). (The results are published in Journal of Instrumentation.)

8. Integration of the MuCh simulation software with the `CbmRoot` simulation framework to generate a free running data stream similar to that expected from the real experiment.

9. Developed noise generation framework for MuCh system such that the treatment of thermal, uncorrelated noise has been included in the simulated data stream.

Bibliography

[1] A. Kiseleva, P. P. Bhaduri, S. Chattopadhyay, et al. Di-muon measurements with the CBM experiment at FAIR. https://doi.org/10.1007/s12648-011-0043-5. 1, 13

[2] P. P. Bhaduri. Charmonium Production and Detection in High Energy Nuclear Collisions at FAIR. http://www.hbni.ac.in/students/dsp_ths.html?nm=phys/PHYS04201104001.pdf. 1, 17

[3] The CBM Experiment. https://www.cbm.gsi.de/. 1, 4

[4] The Facility for Antiproton and Ion Rsearch FAIR. https://fair-center.eu/. 1, 4

[5] Ablyazimov T et al. Challenges in QCD matter physics -- The scientific programme of the Compressed Baryonic Matter experiment at FAIR. *The European Physical Journal A*, 53(3):60, 2017. DOI:10.1140/epja/i2017-12248-y. 1, 13

[6] V. Singhal, S. Chattopadhyay, and V. Friese. Investigation of heterogeneous computing platforms for real-time data analysis in the cbm experiment. *Computer Physics Communications*, 253:107190, 2020. DOI:10.1016/j.cpc.2020.107190. 2, 14, 54

[7] Subhasis Chattopadhyay et al., editors. *Technical Design Report for the CBM : Muon Chambers (MuCh)*. GSI, 2015. https://repository.gsi.de/record/161297. 3, 18, 76, 78, 87

[8] GSI Accelerator. https://www.gsi.de/en/researchaccelerators/accelerator_facility/ring_accelerator, June 2020. 4, 7

[9] J. de Cuveland and V. Lindenstruth (for the CBM collaboration). A first-level event selector for the CBM experiment at FAIR. *Journal of Physics: Conference Series*, 331(2):022006, dec 2011. DOI: 10.1088/1742-6596/331/2/022006. 7, 43

[10] Computing Farm named Green IT Cube. https://www.gsi.de/en/work/it/data_center.htm, November 2020. 7

[11] Javier Diaz et al. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1369–1386, 2012. DOI:10.1109/TPDS.2011.308. 8, 34

[12] V Friese and for the CBM Collaboration. The high-rate data challenge: computing for the CBM experiment. *Journal of Physics: Conference Series*, 898:112003, oct 2017. DOI:10.1088/1742-6596/898/11/112003. 10, 14, 39, 100

[13] Volker Friese. Simulation and reconstruction of free-streaming data in CBM. *Journal of Physics: Conference Series*, 331(3):032008, dec 2011. DOI:10.1088/1742-6596/331/3/032008. 10

[14] CbmRoot: Simulation and Analysis framework for CBM Experiment. https://indico.cern.ch/event/408139/contributions/979676/attachments/815543/1117470/cbmroot_CHEP06.pdf. 10, 16

[15] FairRoot. https://fairroot.gsi.de/, Jan. 2021. 10, 16

[16] Alexander Malakhov and Alexey Shabunov, editors. *Technical Design Report for the CBM Superconducting Dipole Magnet*. GSI, Darmstadt, 2013. https://repository.gsi.de/record/109025. 14

[17] R Brun, F Bruyant, M Maire, A C McPherson, and P Zanarini. *GEANT 3: user's guide Geant 3.10, Geant 3.11; rev. version*. CERN, Geneva, 1987. https://cds.cern.ch/record/1119728. 16

[18] GEANT 4. http://geant4.web.cern.ch, April 2019. 16

[19] O. Singh et al. Description of the cbm-much geometry in cbmroot. *CBM Progress Report*, page 140, 2017. DOI:10.15120/GSI-2018-00485. 18, 63

[20] S.A. Bass et al. Microscopic models for ultrarelativistic heavy ion collisions. *Progress in Particle and Nuclear Physics*, 41:255–369, 1998. https://www.sciencedirect.com/science/article/pii/S0146641098000581. 20, 21

[21] I Fröhlich, T Galatyuk, R Holzmann, J Markert, B Ramstein, P Salabura, and J Stroth. Design of the pluto event generator. *Journal of Physics: Conference Series*, 219(3):032039, apr 2010. DOI:10.1088/1742-6596/219/3/032039. 20, 21

[22] V. Singhal et al. Real time data analysis using GPU for high energy physics experiments. http://www.sympnp.org/proceedings/57/G57.pdf, 2012. 20, 46

[23] Valentina Akishina and Ivan Kisel. Time-based cellular automaton track finder for the CBM experiment. *Journal of Physics: Conference Series*, 599:012024, apr 2015. DOI:10.1088/1742-6596/599/1/012024. 24

[24] Thematic CERN School of Computing. *Collaboration with the University of Split, Croatia*, May 2015. 25

[25] Priya Sen and Vikas Singhal. Event selection for much of cbm experiment using gpu computing. In *2015 Annual IEEE India Conference (INDICON)*, pages 1–5, 2015. DOI:10.1109/INDICON.2015.7443569. 26

[26] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. L. Wang. Heterogeneous Computing: Challenges and Opportunities. *IEEE Computer*, 1993. DOI:0018-9162/93/0600-0018503. 32

[27] I. Ekmecic, I. Tartalja, and V. Milutinovic. A taxonomy of heterogeneous computing systems. *Computer*, 28(12):68–70, 1995. DOI:10.1109/2.476202. 32

[28] The Open Group Base Specifications Issue 7. 2018 edition, IEEE Std 1003.1-2017. 34

[29] OpenMP. www.openmp.org/, April 2019. 34

[30] MPI. http://www-unix.mcs.anl.gov/mpi/. 34

[31] W. Gropp S. Huss-Lederman A. Lumsdaine E. Lusk B. Nitzberg W. Saphir and M. Snir. MPI: The Complete Reference, Vol 2: The MPI-2 extensions. *MIT Press, Cambridge MA*, 1998. 34

[32] W. GroppE. Lusk and R. Thakur. Using MPI-2: Advanced Featuresof the Message-Passing Interface. *MIT Press, Cambridge MA*, 1999. 34

[33] OpenCL. https://www.khronos.org/opencl/, October 2018. 34, 49

[34] CUDA. http://www.nvidia.com/cuda, October 2018. 34

[35] J. Sanders and E. Kandrot. CUDA By Example - An Introductionto General-Purpose GPU Programming. *Addison-Wesley Professional*, 2010. 34, 50

[36] NVIDIA TESLA C2075 Companion Processor. www.nvidia.in/docs/IO/43395/NV-DS-Tesla-C2075.pdf. 39, 40, 51

[37] QUADRO. http://www.nvidia.in/object/product-quadro-4000-in.html, October 2018. 39, 51

[38] Roberto Di Salvo and Carmelo Pino. Image and video processing on cuda: State of the art and future directions. page 60 66, 2011. DOI: 10.5555/2074857.2074871. 41

[39] D. Kim. Evaluation of the Performance of GPU Global Memory Coalescing. http://www.jmest.org/wp-content/uploads/JMESTN42352129.pdf, 2017. 43, 44

[40] Minquan Fang, Jianbin Fang, Weimin Zhang, Haifang Zhou, Jianxing Liao, and Yuangang Wang. Benchmarking the gpu memory at the warp level. *Parallel Computing*, 71:23–41, 2018. DOI:10.1016/j.parco.2017.11.003. 46

[41] G Lamanna et al. GPUs for real-time processing in HEP trigger systems. *Journal of Physics: Conference Series*, 513(1):012017, jun 2014. DOI:10.1088/1742-6596/513/1/012017. 47

[42] A. Jain and V. Singhal. Investigations of High Throughput Computing for Event Selection Process of MuCh@CBM. https://doi.org/10.2139/ssrn.3366339, 2019. 49, 52

[43] Kamran Karimi, Neil G. Dickson, and Firas Hamze. A performance comparison of cuda and opencl, 2011. arxiv: 1005.2581. 50

[44] Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A comprehensive performance comparison of cuda and opencl. In *2011 International Conference on Parallel Processing*, pages 216–225, 2011. DOI:10.1109/ICPP.2011.45. 50

[45] P. B. Hansen. Studies in Computational Science: Parallel Programming Paradigms. 1995. 52

[46] M. J. Quinn. Parallel Computing: Theory and Practice. 1994. 52

[47] I. Hrivnacova et al. The virtual monte carlo. *CoRR*, 2003. arxiv: cs.SE/0306005. 63

[48] Fabio Sauli. The gas electron multiplier (GEM): Operating principles and applications. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 805:2–24, 2016. DOI:10.1016/j.nima.2015.07.060. 65

[49] *Selected Topics in Nuclear Electronics*. Number 363 in TECDOC Series. 1985. 67, 71

[50] Krzysztof Kasinski et al. STS-XYTER, a high count-rate self-triggering silicon strip detector readout IC for high resolution time and energy measurements. pages 1-6, 2014. DOI:10.1109/NSSMIC.2014.7431048. 67

[51] CbmRoot 2016. https://subversion.gsi.de/cbmsoft/cbmroot/release/JUN16/. 68, 70

[52] CbmRoot 2018. https://subversion.gsi.de/cbmsoft/cbmroot/release/OCT18/. 68

[53] V. Singhal, S. Chatterjee, S. Chattopadhyay, and V. Friese. Development and implementation of a time-based signal generation scheme for the Muon Chamber simulation of the CBM experiment at FAIR. *Journal of Instrumentation.*, 16, August 2021. DOI:10.1088/1748-0221/16/08/p08043. 68

[54] K. Kasinski, A. Rodriguez-Rodriguez, J. Lehnert, W. Zubrzycka, R. Szczygiel, P. Otfinowski, R. Kleczek, and C.J. Schmidt. Characterization of the STS/MUCH-XYTER2, a 128-channel time and amplitude measurement IC for gas and silicon microstrip sensors. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 908:225–235, 2018. https://www.sciencedirect.com/science/article/pii/S0168900218310349. 69, 71, 73, 79

[55] Rama Prasad Adak et al. Performance of a large size triple gem detector at high particle rate for the cbm experiment at fair. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 846:29–35, 2017. https://www.sciencedirect.com/science/article/pii/S0168900216312530. 78

[56] Lebedev, S. et al. Event reconstruction of free-streaming data for the rich detector in the cbm experiment. *EPJ Web Conf.*, 214:01043, 2019. DOI:10.1051/epjconf/201921401043. 79, 86, 98

[57] V. Singhal, S. Chattopadhyay, and V. Friese. Event-building process from free-streaming data. *GSI Scientific Report 2014 and CBM Progress Report 2014.*, 2014. ISBN: 978-3-9815227-1-6. 86, 98

[58] V. Singhal, S. Chattopadhyay, P. Ganguly, K. Sarkar, A. Tewary, A. Chakrabarti, and V. Friese. Parallelisation of cluster and hit Finding for the CBM-MUCH. *CBM Progress Report*, 2017. DOI:10.15120/GSI-2018-00485. 86

[59] J. Lehnert, A.P. Byszuk, D. Emschermann, K. Kasinski, W.F.J. Müller, C.J. Schmidt, R. Szczygiel, and W.M. Zabolotny. GBT based readout in the CBM experiment. *Journal of Instrumentation*, 12(02):C02061–C02061, feb 2017. DOI:10.1088/1748-0221/12/02/c02061. 87

[60] Hybrid Programming with OpenMP and MPI. Technical Report 18.337J, Massachusetts Institute of Technology 2009. 98

[61] Dana Jacobsen, Julien Thibault, and Inanc Senocak. *An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters*. DOI: 10.2514/6.2010-522. 98

[62] Chao-Tung Yang, Chih-Lin Huang, and Cheng-Fang Lin. Hybrid cuda, openmp, and mpi parallel programming on multicore gpu clusters. *Computer Physics Communications*, 182(1):266–269, 2011. DOI: 10.1016/j.cpc.2010.06.035. 98