Universitatea
Transilvania
din Brașov
FACULTATEA DE INGINERIE ELECTRICĂ
ȘI ȘTIINȚA CALCULATOARELOR

PROIECT DE DISERTAȚIE

Conducător științific:
Prof. dr. ing. Mihai IVANOVICI
Prof. dr. fiz. Mihai PETROVICI

Absolvent:
Adriana-Georgiana NAN

BRAȘOV, 2022

Universitatea Transilvania din Brașov
FACULTATEA DE INGINERIE ELECTRICĂ ȘI ȘTIINȚA CALCULATOARELOR

Departamentul Electronică și calculatoare
Programul de studii Sisteme electronice și de comunicații integrate - RCD

NAN Adriana-Georgiana

VISUALISATION OF DATA AND VALIDATION OF RECONSTRUCTION ALGORITHMS USED IN HADRONIC PHYSICS EXPERIMENTS

VIZUALIZAREA DATELOR ȘI VALIDAREA ALGORITMILOR DE RECONSTRUCȚIE FOLOSIȚI ÎN CADRUL EXPERIMENTELOR DE FIZICĂ HADRONICĂ

Table of contents / Cuprins

Acknowledgements

This endeavor would not have been possible without the generosity of prof. Mihai Petrovici who has given me the opportunity to be part of his team. I thank him for all his trust and kindness. I would also like to thank the entire staff at the Hadronic Physics Department of The National Institute for Physics and Nuclear Engineering, for the warmth with which they have welcomed me to the team. I am deeply indebted to dr. Alexandru Bercuci for providing insightful knowledge and believing in me. His professional advise, enthusiasm and friendship have meant a lot. My heartfelt thanks and appreciation goes to dr. Laura Rădulescu for her friendship, continuous care and support. Last but not least, I would like to express my deepest gratitude to my primary supervisor, prof. Mihai Ivanovici, for his patience, constant support, encouragements and guidance throughout the research and writing of this thesis.

Abstract

The present thesis is based on the work carried out for the Compressed Baryonic Matter experiment (CBM) and in particular for the TRD-2D detector and could not have been possible without the full support of the Hadronic Physics Department (HPD) at the National Institute for Physics and Nuclear Engineering "Horia Hulubei" (NIPNE), Măgurele, Romania.

The CBM is a hadronic physics experiment which will be one of the four main experiments (together with NUSTAR, PANDA and APPA) at the Facility for Antiproton and Ion Research (FAIR) - a particle accelerator that is currently under construction at the Institute for the study of heavy ions - "Gesellschaft fur Schwerionenforschung" (GSI) in Darmstadt, Germany. The experiment is an international collaboration of scientists, engineers and students from over 12 countries. FAIR enables scientists to recreate, in the laboratory, the same conditions of temperature, density and pressure happening in stellar explosions. This is achieved by bombarding small samples of matter with particles, thus creating cosmic matter, known as quark-gluon plasma which decays rapidly into known particles (a phenomenon called hadronization). The particles resulting from these interactions are, then, travelling through sets of detectors which measure the position in which the hits occur and the energy/charge of the particle. By corroborating the data measured by the detectors, scientists can determine what kind of particles were created and their exact time of creation.

For the time being, a prototype of the experiment - the mini-CBM (mCBM) - is being used for testing and fine-tuning the detectors, as well as acquiring data at lower interaction rates than the CBM will achieve. The experimental setup for CBM (and mCBM) comprises a number of detectors from which hits will be corroborated, leading to the identification of rare probes. In the CBM experiment a large number of observables will be created, among them being some extremely rare ones which may appear once in every one million collisions, hence the term "rare probes".

The TRD-2D detector was designed and built for the mCBM by the Hadronic Physics Department (HPD) of the National Institute for Physics and Nuclear Engineering (NIPNE), Măgurele, Romania and is used for identification of electrons but also for track reconstruction.

My work has concentrated on running simulations of the experiment (with a focus on the TRD-2D detector), comparisons between data taken from the mCBM and simulation results, reconstruction of hits and Quality Assurance assessment (QA) which is much needed in the process of online selection[1]. In essence, it can be argued that all work I have been involved in,

[1]Because of the high rate at which the experiment runs, there is an abundance of data - in the order of GB/s for mCBM, and ∼1TB/s for CBM - hence, a sturdy selection has to be made of relevant data. The unneeded data can be discarded right away, thus saving space on disk and bandwidth

is revolving around processing and analysis of big data.

The tool that is used by all members of the CBM collaboration to create simulations and write the programs used for analysis of simulations and data, is called ***cbmRoot***. It is a framework written in C++, derived from the ***root*** framework created at CERN, which is able to handle very large files of data. The cbmRoot framework provides a set of scripts (called ***macros***) which can be used for creating the simulations right off the bat, provided the correct geometry/setup is given. It also provides a graphical interface for the user and accepts command line scripting.

In my work, I have started by generating simulations for different geometries and interaction rates (frequencies from 10kHz to 10MHz). Also, all simulations were run both event-based (with an event being the collision between two nuclei) and time-based (a time frame is determined - for instance 10ms, in which multiple events can occur and are captured in the simulation). This latter approach is more accurately reproducing what is happening in the experiment, and is thus the mode that I have used most often in my simulations, even though it is harder to interpret the results as so-called event-pileups can occur, in which the time between events is so short that individual hits cannot be distinguished.

After generating the simulations, I have moved on to running the event reconstruction macros and checking that they all run without issues, then creating my own macro to match the Monte-Carlo points[2], to the reconstructed hits. The idea in doing these checks was to have one macro that would run error-less on both simulations and on real data. Finally I have worked on a Quality Assurance macro, which should help in determining which data is useful and should be stored, with the end goal to reduce the size of recorded data, in order to save storage space on disk as well as bandwidth. Information about the experiment, the simulations, scripts I have created and the results of my work, will be detailed in the following chapters of this thesis.

[2]"Monte Carlo Simulation (or Method) is a probabilistic numerical technique used to estimate the outcome of a given, uncertain (stochastic) process. This means it's a method for simulating events that cannot be modelled implicitly. This is usually a case when we have a random variables in our processes"[16].There are multiple MC-event generators created for physics experiments; cbmRoot uses GEANT4.

Rezumat

Lucrarea de față este bazată pe munca realizată pentru experimentul de fizică hadronică intitulat Compressed Baryonic Matter (CBM) și în special pentru detectorul TRD-2D, iar realizarea ei nu ar fi fost posibilă fără sprijinul Departamentului de Fizică Hadronică de la Institutul Național de Fizică și Inginerie Nucleară "Horia Hulubei", Măgurele, România.

CBM este unul dintre cele patru experimente de fizică hadronică (alături de NUSTAR, PANDA și APPA) ce vor fi realizate la Centrul pentru cercetarea antiprotonilor și ionilor - Facility for Antiproton and Ion Research (FAIR) - un accelerator de particule care este momentan în construcție la Institutul pentru cercetarea ionilor grei - "Gesellschaft für Schwerionenforschung" (GSI) în Darmstadt, Germania. Experimentul este parte a unei colaborări internaționale, ce include fizicieni, ingineri și studenți din peste 12 țări. Acceleratorul FAIR le permite cercetătorilor să obțină, în laborator, aceleași condiții (de temperatură, presiune, densitate) ce se regăsesc în timpul unei explozii stelare, prin bombardarea unor nuclee cu diferite particule. Aceste coliziuni creează materie cosmică (de dimensiuni foarte mici și cu o durată de viață extrem de scurtă), materie care este cunoscută sub numele de plasmă quark-gluon și care se transformă foarte repede în particule cunoscute (fenomen denumit hadronizare). Particulele rezultate din aceste interacțiuni se deplasează printr-o serie de detectori care măsoară locul în care s-a produs hit-ul și energia depusă de fiecare particulă. Prin coroborarea datelor măsurate de detectorii prezenți în experiment, oamenii de știință pot stabili ce fel de particule au rezultat din interacție și timpul exact al creării lor.

În momentul de față, un prototip al experimentului - mini-CBM (mCBM) - este folosit pentru a testa și calibra detectorii și electronica, precum și pentru a prelua date la rate de interacțiune mai mici decât cele la care va ajunge CBM-ul. Experimentul mCBM (și pe viitor CBM) cuprinde o serie de detectori (proiectați și construiți în diferite institute de cercetare și universități din Europa și Asia), ale căror rezultate vor fi coroborate pentru identificarea probelor rare (identificarea particulelor cu o durată de viață extrem de scurtă - de ordinul femto-secundelor, care apar extrem de rar - aproximativ o dată la un milion de coliziuni).

Detectorul TRD-2D (Transition Radiation Detector 2D) a fost proiectat și construit exclusiv în Departamentul de Fizică Hadronică al Institutului Național pentru Fizică și Inginerie Nucleară "Horia Hulubei", Măgurele, România și este unul dintre detectorii principali în experiment, fiind folosit atât pentru identificarea electronilor cât și pentru reconstrucție de traiectorii.

Munca mea în cadrul experimentului CBM s-a concentrat pe realizarea simulărilor (cu accent pe detectorul TRD-2D), realizarea unor comparații între datele reale obținute în cadrul experimentului mCBM și rezultatele obținute din simulări, reconstrucție de ciocniri în detector ("hit-uri") și realizarea unor evaluări din punct de vedere al calității asupra rezultatelor din

simulări. Aceste evaluări sunt necesare in procesul de selecție a datelor relevante pentru experiment; din cauza frecvențelor mari la care lucrează experimentul, se obține o cantitate foarte mare de date - de ordinul GB/s pentru mCBM și ∼1TB/s pentru CBM. De aceea, o selecție riguroasă a datelor relevante este necesară, lucru care va reduce spațiul ocupat pe servere și lățimea de bandă necesară pentru transportul acestora. În esență, se poate considera că munca mea se concentrează în jurul conceptului de "Big Data" - procesare și analiză de volume mari de date.

Pentru procesarea datelor experimentale, dar și pentru realizarea simulărilor și manipularea datelor obținute, toți membrii colaborării CBM utilizează biblioteca ***cbmRoot***. Acesta este un cadru C++ derivat din cadrul ***root*** creat la CERN și capabil să trateze fișiere foarte mari de date. ***cbmRoot*** pune la dispoziția utilizatorului un set de script-uri (denumite ***macro-uri***) care pot fi utilizate pentru a rula simulările pentru experiment (cu posibilitatea de a alege geometria/setup-ul experimental dorit). De asemenea, oferă utilizatorului o interfață grafică și o interfață tip linie de comandă.

În studiul meu, am început prin rularea simulărilor pentru diferite geometrii și frecvențe (de la 10kHz la 10MHz), precum și pentru diferite tipuri de interacție (coliziunea unor nuclee diferite). Simulările au fost rulate în ambele moduri posibile: bazat pe eveniment (unde un eveniment este reprezentat de coliziunea a două nuclee) și bazat pe timp (într-un interval de timp determinat - de exemplu 10ms, se va produce un număr mare de evenimente, care vor fi eșantionate). Acest al doilea mod de rulare este mai apropiat de ceea ce se întâmplă în experimentul real, de aceea am folosit acest mod preponderent în simulările mele, deși rezultatele sunt mult mai dificil de interpretat deoarece există fenomene de tip acumulare de evenimente ("event pileup") care se produc din pricina faptului că timpul între evenimente este atât de mic, încât hit-urile devin indistinctibile.

După generarea simulărilor, am trecut la verificarea macro-urilor de reconstrucție de evenimente, asigurându-mă că ele funcționează fără erori - o serie de macro-uri conținute în cbmRoot au fost create pentru a funcționa pe date experimentale, dar nu și pe simulări și vice versa. Pentru a avea însă o imagine clară asupra corelației între date și simulări, avem nevoie să putem rula același macro, pe ambele tipuri de date. Am continuat prin crearea propriului macro pentru corelarea punctelor Monte-Carlo[3] cu hit-urile reconstruite. În cele din urmă, am lucrat la macro-ul de QA (asigurarea calității), care ajută în determinarea relevanței anumitor date obținute din experiment, astfel încât datele irelevante să poată fi filtrate, reducând astfel spațiul de stocare. În capitolele ce urmează, voi vorbi în detaliu despre experiment, despre simulări și necesitatea lor, metodologia de lucru, script-urile create de mine și rezultatele obținute.

[3]"Metoda/Simularea Monte Carlo este o tehnică matematică probabilistică, utilizată pentru a estima rezultatele unui proces stohastic. Altfel spus, este o metodă prin care se pot simula evenimente ce nu pot fi modelate implicit, cum este cazul proceselor ce depind de variabile aleatorii."[16]. Exista o multitudine de generatorare de evenimente Monte-Carlo, create pentru experimentele de fizica; cbmRoot utilizeaza generatoarele GEANT3 și GEANT4.

Acronyms

FAIR	-	Facility for Antiproton and Ion Research
GSI	-	Gesellschaft fur Schwerionenforschung
NIPNE	-	National Institute for Physics and Nuclear Engineering
HPD	-	Hadronic Physics Department
NIHAM	-	Nuclear Interactions and Hadronic Matter Centre of Excellence
CBM	-	Compressed Baryonic Matter
TRD	-	Transition Radiation Detector
STS	-	Silicon Tracking System
ToF	-	Time of Flight Detector
RICH	-	Ring Imaging Cherenkov Detector
GeV	-	Giga-electron volt
AGeV	-	Giga-electron volt / nucleon
MC points	-	Monte-Carlo points
FASP	-	Fast Analog Signal Processor
FEE	-	Front End Electronics
UrQMD	-	Ultra relativistic Quantum Molecular Dynamic
GEANT	-	GEometry ANd Tracking

Chapter 1

Introduction

In the present chapter I will describe the concept of free running data acquisition and the Transition Radiation Detector 2D (TRD-2D), involved in the Compressed Baryonic Matter experiment (CBM) at the Facility for Antiproton and Ion Research (FAIR) in Darmstadt, Germany. However, in order to give you a clear image on the context and the importance of this work, I will provide some more general information about the role of particle accelerators, the CBM experiment, the detector and the role of simulations in the entire chain.

1.1 Investigating sub-atomic matter with particle accelerators

All matter in the Universe is made up of atoms, which are composed of a nucleus and surrounding electrons. The nucleus of an atom is made up of protons and neutrons (both known in the world of physics as baryons) which in turn are composed of quarks (the two types being up and down quark) which are bound together by gluons (also known as the strong force). In the natural world, under normal circumstances, all quarks are subjected to this strong force which glues them together.

In the first moments after the Big Bang, however, the resulting matter is thought to have been 1 million times hotter than the center of our sun, with all subatomic particles floating around, unbound (this deconfined matter being called Quark-Gluon Plasma). The same kind of matter is thought to make up Neutron stars (tiny stars of only 10-16km in diameter, but with huge masses of up to 2 times the mass of the sun). This deconfined matter is short-lived once the conditions in which it was formed change and will very quickly decay/transition into hadrons - meaning that the gluons will once again bind quarks together forming different types of particles (see Figure 1.1).

In order to study this Quark-Gluon Plasma (QGP) and calculate the moment and conditions in which hadronization begins, scientists need to recreate these conditions of high temperatures, density and pressure and for that, particle accelerators are used.

While experiments such as the Large Hadron Collider (LHC) at CERN, Geneva and the Relativistic Heavy Ion Collider (RHIC) at Brookhaven National Laboratory in Long Island, New York, USA are exploring QGP resulted from interactions at very high speeds (bare atomic nuclei are collided at such high speeds that protons and neutrons are effectively melted and Quark-Gluon Plasma is obtained), others such as the Compressed Baryonic Matter Experiment

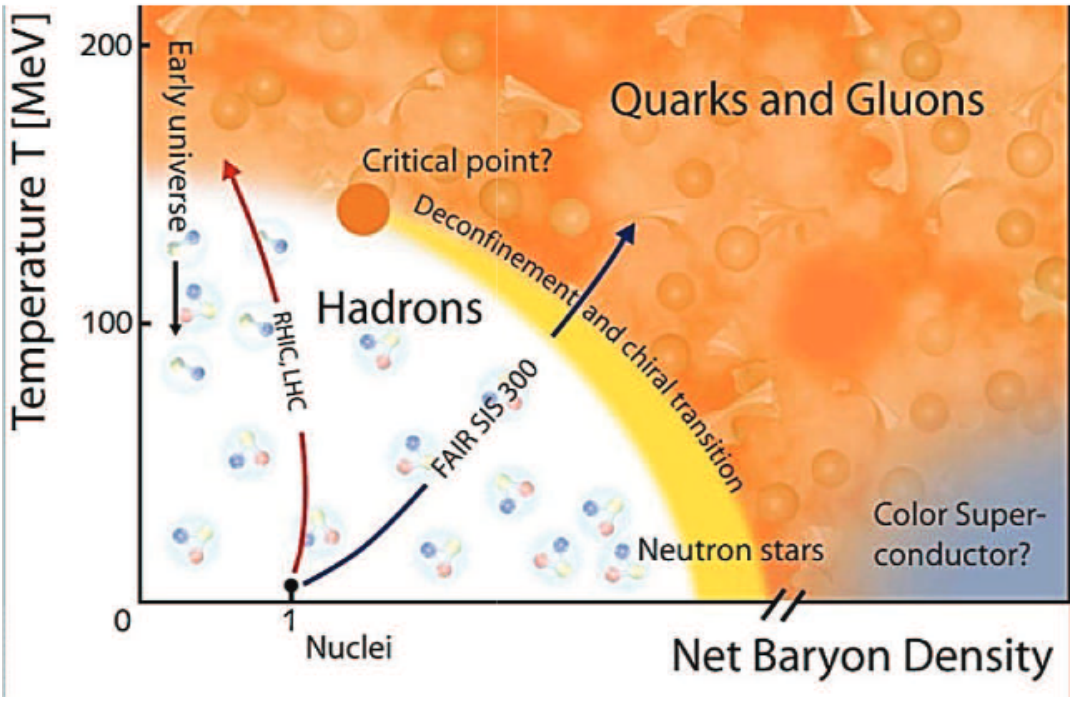


Figure 1.1: Quark-Gluon Plasma Transition Diagram [7]. Exploring the QGP Diagram - at higher temperature, the conditions created are similar to the early Universe, just after the Big Bang (two facilities where matter is explored in these conditions are RHIC and LHC) ; at high baryon densities, the conditions are similar to the ones found in Neutron Stars and these conditions are simulated and analyzed at FAIR.

(CBM) at the Facility for Antiproton and Ion Research (FAIR), Darmstadt, Germany are exploring the QGP at very high baryonic densities (similar to the conditions existing in Neutron star cores, where the neutrons are subjected to such a high pressure that Quark-Gluon Plasma is formed).

At CBM ("The Universe in the Lab"), the installation used to accelerate particles will be a SIS100 accelerator which delivers moderate collision energies for fixed-target operation, in the energy range of 2.7-4.9 GeV, but at very high interaction rate *"which leads to very baryon-dense collision zones approaching anticipated neutron star core densities."*[15]

1.1.1 The Monte Carlo models to describe nuclear interactions

In experimental physics, Monte Carlo (MC) models are used to describe the particles resulted from the collision of nuclei, as well as their trajectories through the experimental setup and the detectors' response to the interaction of particles with them. Some of the advantages in using the Monte Carlo method for modelling nuclear interactions are :

- *"Theory driven and data benchmarked*
- *Naturally extended to materials/energies where no data are available*
- *Natural accounting for non-measured secondaries*
- *Event-by-event description, with full correlations and exact energy conservation, provided*

analogue, self-consistent models are used both for interaction modelling and particle transport

- *Description of complex geometries, including time-varying problems*
- *Transport in electric and magnetic fields*
- *Fully three-dimensional calculations*
- *Variance reduction by biasing techniques"[3]*

The transport engines used in CBM to simulate *"the passage of particles through matter"*[13] using Monte Carlo methods are GEANT3 and GEANT4 ("GEometry ANd Tracking"). The software was created at CERN and its first release dates back to 1974. The trajectory of the particle and all the major interactions with the detector systems are stored in a binary "root" file which can be accessed and evaluated [5] (or visualised), but can also serve as input for further processing.

Another Monte-Carlo model which is used in High Energy Particle Physics is an event generator (simulating the initial nucleus-nucleus interaction) called Ultra relativistic Quantum Molecular Dynamics (UrQMD).

1.2 The CBM experiment @ FAIR

1.2.1 Investigating the compressed matter state and accessing rare probes of the compressed matter

In Neutron stars as well as *"during stellar explosions and collisions, matter is subject to extreme conditions such as very high temperatures, pressures, and densities"* [14], which turn it into Quark-Gluon plasma. In order to understand the phenomenon, we could make a simple analogy to the most common element on Earth - water. At low temperatures, water is found in the form of ice (solid water); apply heat to it and it will turn into liquid, but keep heating it and it will eventually reach boiling point, turning it into steam (gaseous state). This process is reversible and you can obtain liquid from gas by cooling it down. The Quark-Gluon Plasma is very similar, it gets produced by overheating matter (either by using high temperatures or high pressure/densities) and will seize to exist once it cools down.

At FAIR, high densities will be produced for short periods of time approx. 5-10 fm/c (Femtometers at the speed of light in vacuum), in which new particles are created. *"The experimental challenge is to identify these partly very rarely produced diagnostic probes and to measure their yield and momentum with high precision and unprecedented statistics"* [20]. Some of the observables which will be produced by CBM (such as open and hidden charm) are expected to have a multiplicity of one in every 1 million collisions, and that is why it's worth mentioning that the CBM will run at an unprecedented reaction rate of 10MHz, whereas other heavy-ion experiments are run at rates well under 100kHz (see Figure 1.2).

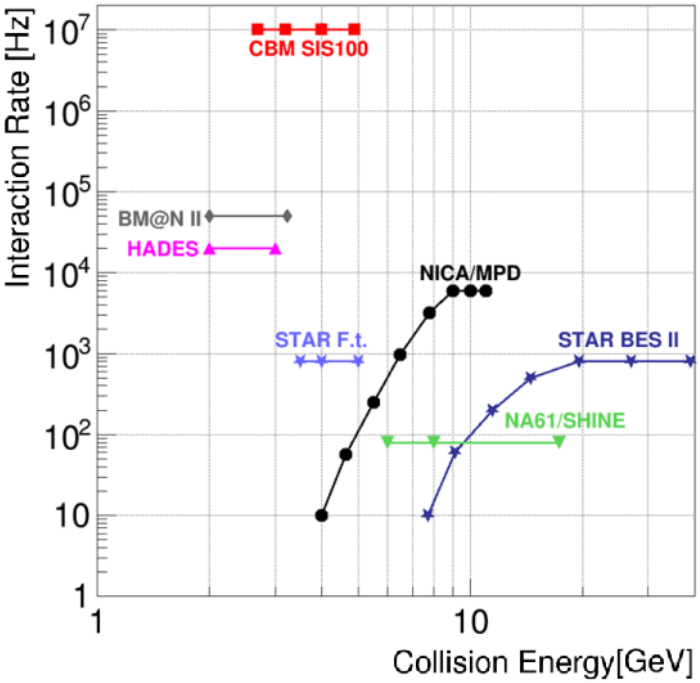


Figure 1.2: CBM interaction rates [12]. The CBM experiment will be working at the highest interaction rates of all existing experiments in it's category - experiments exploring matter at high density. These high data rates give CBM its precision.

1.2.2 Experimental setup

The CBM experiment will be located in a concrete cave, connected directly to the SIS100 accelerator (see Figure 1.3). All detectors in the experiment are required to be fast and radiation resistant, in order to perform at such high interaction rates.

The particles resulting from the collisions will first pass through a Micro Vertex Detector (MVD), then through the Silicon Tracking System (STS) which is designed to determine their trajectories and momenta with high precision. The Muon Chamber and Ring Imaging Cherenkov detectors which follow in the setup, are interchangeable (so only one of the two will be present in the experiment at any given time). Next, the Transition Radiation Detectors (TRD-1D and TRD-2D) are responsible for identifying and tracking particles at high momenta. The Time of Flight Detector (ToF) then identifies particles by their time of flight. The Elecromagnetic Calorimeter (ECAL) used for identification of photons and Projectile Spectator Detector (PSD) for event characterization follow.

Figure 1.4 shows all detectors and their position in the final experiment. Each one of these detectors is designed and implemented in a different group, institution and/or country, thus making the coordination between them and the fact that they are working together for a common end-goal, a truly remarkable effort. TRD-1D and TRD-2D are not clearly distinguished in the figure, since they both have the same task in the experiment, they are however two very different detectors in the way they identify and track particles. So even though they are viewed as a group of detectors, they are designed to be built, by different entities, the TRD-1D in Germany and the TRD-2D at the National Institute for Physics and Nuclear Engineering in Măgurele, Romania.

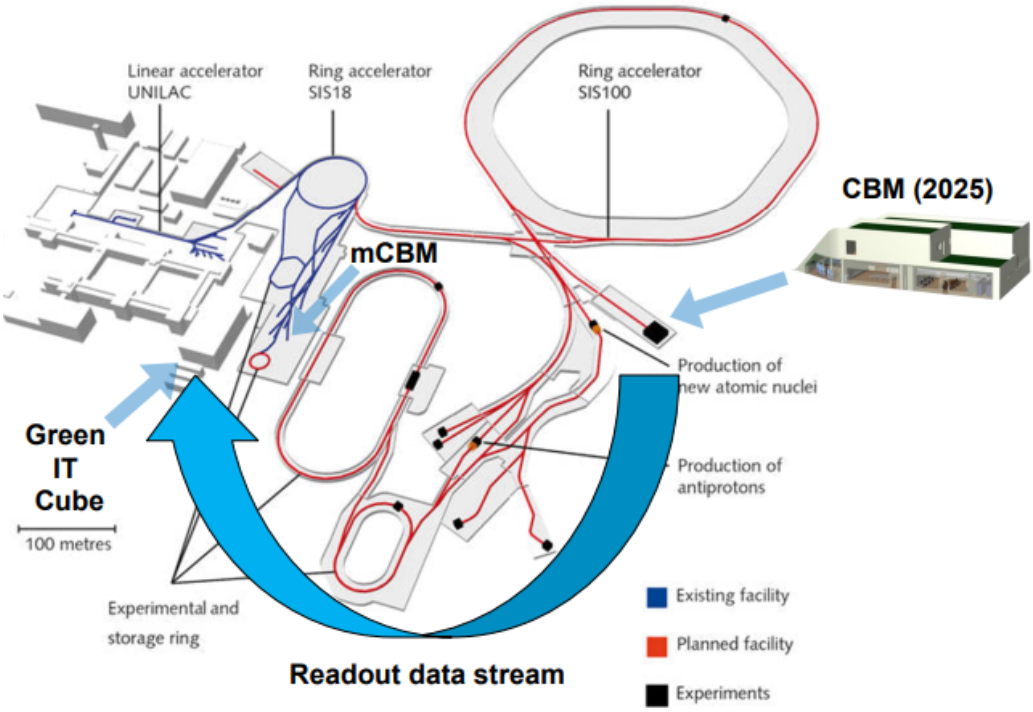


Figure 1.3: FAIR site with CBM and mCBM [11].

1.3 The phase 0 CBM experiment: mCBM @ SIS18

In preparation of the CBM experiment, for which the first beam is planned for 2026, a phase 0 experiment - the mini-CBM (mCBM) has already been implemented. This phase 0 experiment is linked directly to the SIS18 synchrotron, which will act as a pre-accelerator for SIS100. All detectors which were proposed for the CBM experiment, are present in the mCBM whose primary goal is to test and optimize the complex interplay of all different subdetectors under realistic experiment conditions [17]. The current setup at the mCBM is presented in Figure 1.5, in which the two TRDs (1D and 2D) are clearly marked and also in Figure 1.6 which shows the actual setup present in the mCBM cave at this moment.

1.3.1 TRD-2D implementation

There are two TRD-2D modules installed at the mCBM in front of the TRD-1D - one of which is a hybrid readout chamber with both Self-triggered Pulse Amplification and Digitization ASIC (SPADIC) and Fast Analog Signal Processor (FASP) [6] while the Front End Electronics (FEE) for the other is based solely on FASP. The detector I will be referring to as TRD-2D in this thesis is the one based only on FASP (given that it outperforms the hybrid detector and will be the version chosen for CBM).

A simplified description of the way the TRD works would be the following: charged particles which are produced in the interaction between two nuclei, will enter the TRD and excite the gas that is contained in the detector, resulting in electrical signals which are then detected by the FEE. Based on the position and amplitude of these signals and based on the readings from the other detectors in the experiment, calculations can then be made for identifying the

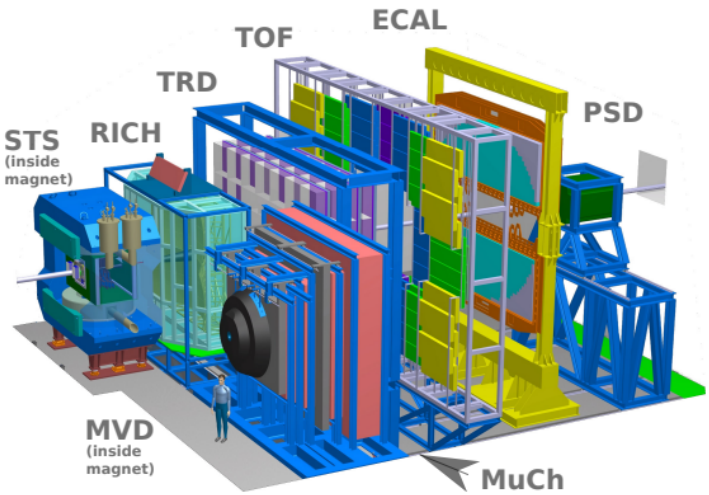


Figure 1.4: CBM - detectors setup [9]

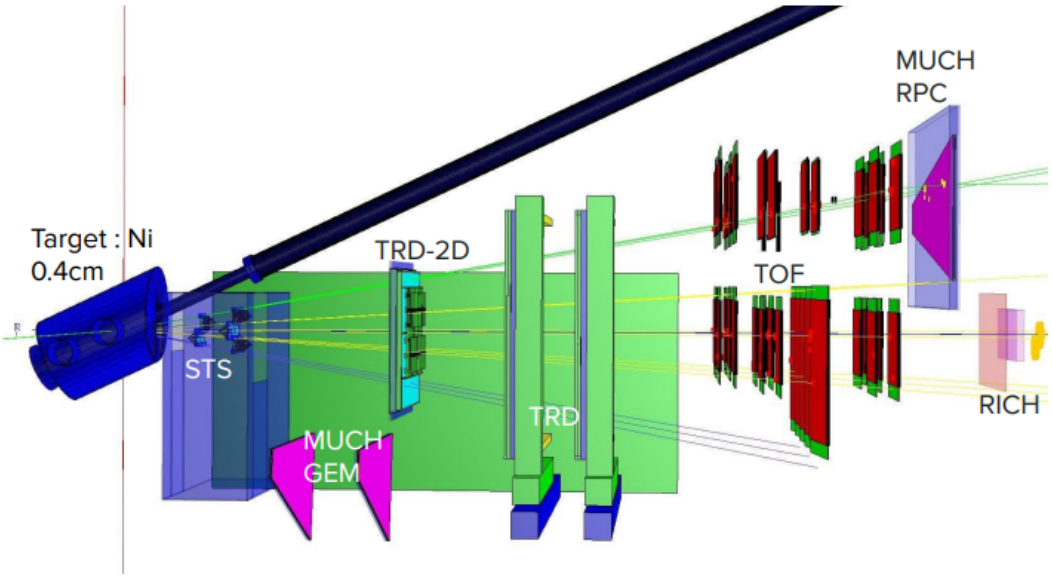


Figure 1.5: mCBM setup [19].

particles and tracking their trajectories through the experimental setup.

1.3.2 Free running data acquisition

The CBM experiment is designed *"to measure particles resulting from collisions of nuclei at an unprecedented interaction rate of up to 10MHz, but in order to achieve this high rate capability, the detectors at CBM will be readout via a triggerless-streaming data acquisition system, transporting data with a bandwidth of up to 1 TB/s to a large scale computer farm for event reconstruction and first level event selection"* [21].

"Because of the high interaction rates, which do not tolerate a significant trigger latency, and because of the complex trigger signatures which are almost impossible to implement in hardware, CBM chose a readout concept based on self-triggered, free-streaming front-end electronics, which deliver time-stamped data messages to the data acquisition system on activation of a read-

[1]Courtesy of Dr. Alexandru Bercuci - HPD, NIPNE, Măgurele, Romania

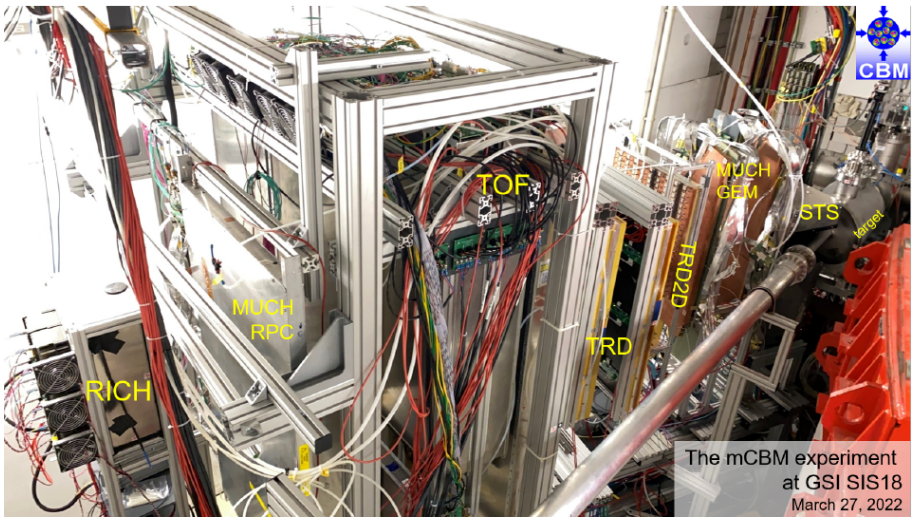


Figure 1.6: Current setup in mCBM cave. [1]The image shows the current setup in the mCBM cave, with the collision point on the right, then from the right to the left: the STS, MUCH, TRD-2D, TRD-1D, ToF, MUCH RPC and RICH detectors.

out channel above a predefined threshold" [4]. Existing experiments work with a hardware implemented triggering system which means that the fastest detector in the experiment is chosen as the trigger and once it detects a particle interaction, it will signal to all the other detectors that they should start listening as well (it effectively sends an electrical signal waking all the other detectors). In a hardware triggered experiment, event selection is done a priori, since the event that will be analyzed, are all decided by the triggering detector. The high interaction rate of the CBM experiment, makes it impossible to implement such a trigger (CBM is a fixed-target experiment in which one nucleus is effectively bombarded with matter). Thus, every detector will measure any hit it perceives and the event selection is done after the data is collected from all the detectors.

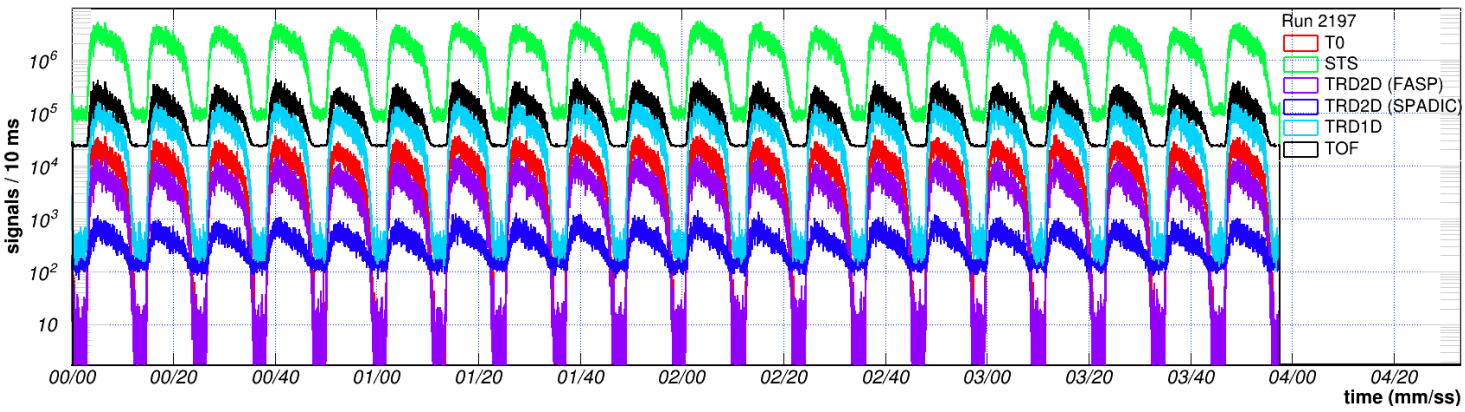


Figure 1.7: Data rates mCBM. [2]Data rates obtained in mCBM run 2197. Note the fact that the TRD-2D has a very good S/N ratio - it is completely quiet on beam-off, while other detectors like ToF and STS register more noise on beam-off than the data registered by the TRD-2D on beam-on and a lower data rate - a factor of 10 rise, while TRD-2D has a factor of over 10^3.

Figure 1.7 shows the high data rates which were obtained at mCBM (in number of signals/10ms), providing a glimpse into the enormous amount of data we will obtain in the CBM

[2]Courtesy of Dr. Alexandru Bercuci - HPD, NIPNE, Măgurele, Romania

runs. The beam-on / beam-off events are clearly visible, as well as the fact that for both TRD-2D and the T0 detectors the number of signals goes down to zero on beam-off, indicating a very good Signal to Noise Ratio. At beam-on, the number of signals reported by the TRD-2D is of 10^4, with T0 indicating only slightly more, which makes the two detectors the most sensitive in the experiment.

It is important to note that, while TRD-2D remains quiet on beam-off and registers a significant rate of data on beam-on, other detectors continue to send data even in beam-off and the rise in data rates on beam-on is relatively small; looking at ToF and STS, it's clear that they produce more noise on beam-off than the volume of data sent by TRD-2D on beam-on. Since the model of data acquisition for CBM is based on free-streaming readouts from all the detectors, the fact that some of them produce such a high level of noise, is not desirable.

Also notice the extreme disproportion of signal to noise measured by the hybrid TRD-2D (here noted as TRD2D SPADIC), which is one of the reasons this type of detector will not be implemented in the CBM experiment.

Chapter 2

Simulations for the TRD-2D

2.1 The software framework of the CBM experiment

The framework used by the members of the CBM collaboration in all activities related to simulations, reconstruction and data analysis is called CbmRoot and is based on FairRoot, a C++ software which is common to all FAIR experiments. FairRoot in turn, is based on the ROOT Analysis Framework, designed and implemented at CERN and commonly used in high energy physics experiments [8].

2.1.1 Simulations, unpacking, reconstruction and tracking

Simulations are extremely important to the experiment, since they are used to model the entire setup, test and optimize algorithms which are designed and deployed for the discovery of rare observables. Figure 2.1 presents an overview of the workflow for the experiment, with a comparison between steps taken in simulations and in processing of data obtained in the data acquisition campaigns.

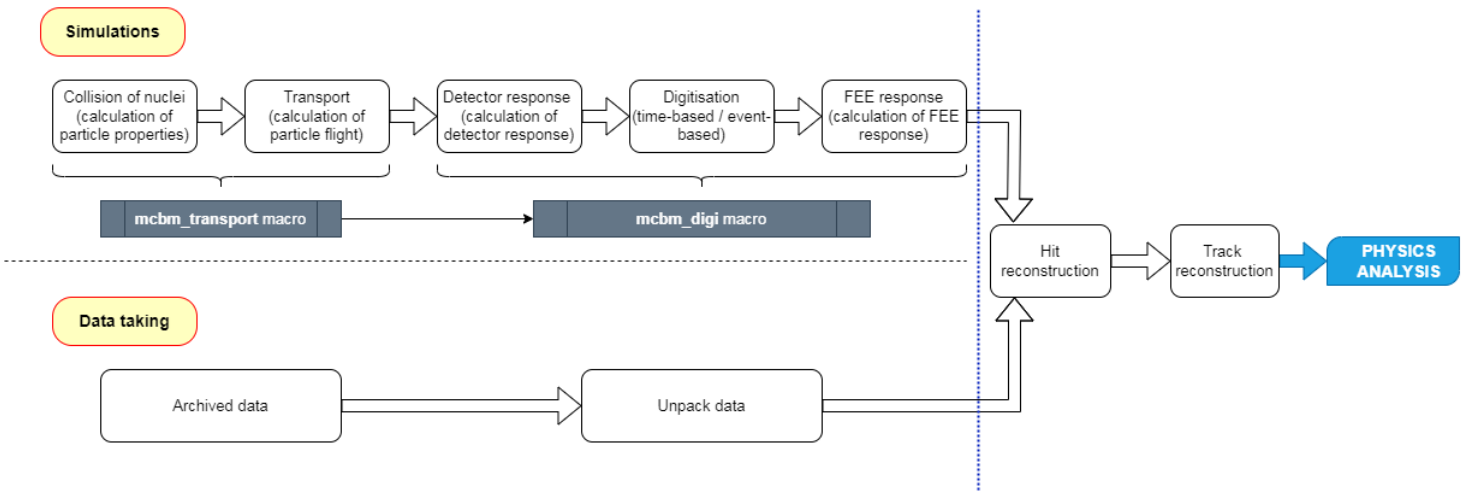


Figure 2.1: Workflow for simulations and data acquisition. The macros used in simulations to mimic collision, transport and detector responses, yield the same data structure obtained in data taking campaigns; this data is then used to create the hit reconstruction and track simulation.

For the first step - determination of the particles resulting from the collision of two nuclei

and of their properties, the most commonly used model in the CBM energy regime is the Ultra Relativistic Quantum Molecular Dynamics (UrQMD) approach (also known as event generator engine). *"The model describes the phenomenology of hadronic interactions at low and intermediate energies"* [5]. The simulations of the initial interaction can be done in two ways:

- central collision - meaning that each particle is hitting the target centrally
- minimum bias - where the particles may or may not hit the target and a hit is not necessarily central

When using central collision we can obtain an order of magnitude more data, the minimum bias model is the one that more accurately describes what is happening in the real experiment. An example of step 1 simulation for Au-Au collision in the mCBM, at an enery of 2.5GeV, is presented in Figure 2.2, in which only a part of the resulting particles are depicted (if we were to allow all particles to show on the figure, we wouldn't be able to see the detectors anymore).

The transport step, which uses Monte Carlo methods to calculate the propagation of the created particles through the experimental setup, is done with the GEANT3 and GEANT4 frameworks. Both of these initial steps are comprised and executed with the *mcbm_transport* macro, already available in cbmRoot.
The response of the detector and Front End Electronics (FEE) - simulated using again GEANT3 / GEANT4, as well as the mapping of interactions to digits, is done in the macro *mcbm_digis*, which can be run in two modes: time-based and event-based. The results are deposited in a 'root' file, which can then be processed further to reconstruct hits and trajectories of particles through the detectors, calculation of vertexes and quality assessment of obtained data.

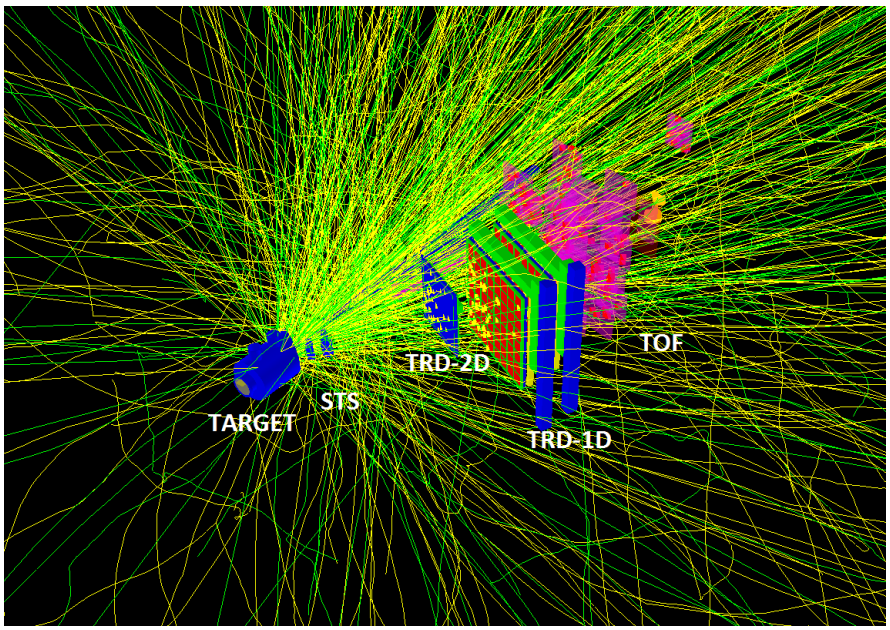


Figure 2.2: Au-Au nuclear interaction simulation - first macro
The image shows a simulation of the abundance of particles resulting from one gold on gold interaction

The steps from simulation, which are generating data using the UrQMD and MC models, are aimed at producing the same kind of digi files as the ones resulting from data taking campaigns.

The data obtained in planned runs of the mCBM experiment is archived in 'tsa' files which need to be unpacked and transformed into 'root' files. This unpacking is done in custom scripts written for each type of detector.

As mentioned in the previous chapter, in the CBM and mCBM, each detector sends its own time-stamped data to the data acquisition system, so in each time interval there are multiple hits coming from different events (with an event being the actual interaction of nuclei in the target). In simulations, the same structure of data has been implemented (see Figure 2.3). Reconstruction of events starts from the reconstruction of hits in each time slice. The reconstruction of hits is done by an intermediary step added to the cbmRoot, in which data coming from the detectors is grouped into clusters. The detectors present in the CBM experiment have a pad structure, with each pad representing a read-out channel. When a particle hits the detector, it's not always hitting a single pad but more often it traverses or fires multiple pads (example in Figure 2.4). All these responses form a cluster of hits, and there are complex algorithms defined to perform this clusterization. In order to narrow down the position of the initial hit, trajectories are calculated, in which multiple detectors are included. Once the hits have been reconstructed, tracking can be performed, to determine the primary vertex (the initial interaction point of particles).

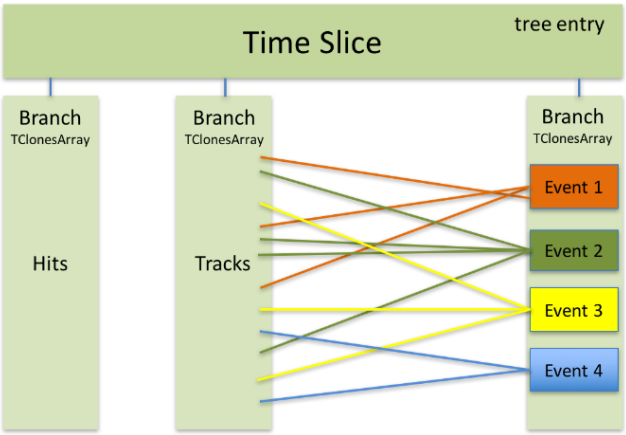


Figure 2.3: Data model for CBM - one tree entry is one timeslice with multiple events [4]

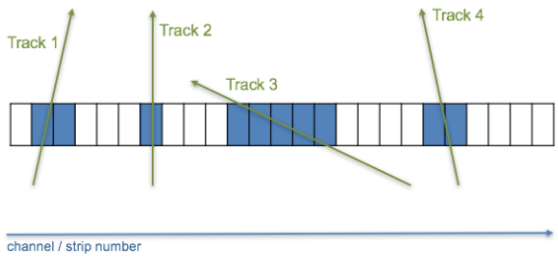


Figure 2.4: Detector's response - clusterization [10].

2.1.2 The cluster finder / hit producer in action on the TRD-2D

The clusterization algorithm is depicted in Figure 2.5, which shows the representation of an interaction in the TRD-2D detector. The data for which these plots are created is taken from a real data run at mCBM. Further on, the front end electronics of the detector will pick up on the electrical signals generated by a particle hit, meaning that not only one readout channel will be reporting the hit, but multiple channels. In order to create clusters of hits, a correlation between time, position and energy of the interaction needs to be computed.

In the first plot, on the left, on the vertical axis the channels which reported the interaction

[1]Courtesy of Dr. Alexandru Bercuci - HPD, NIPNE, Măgurele, Romania

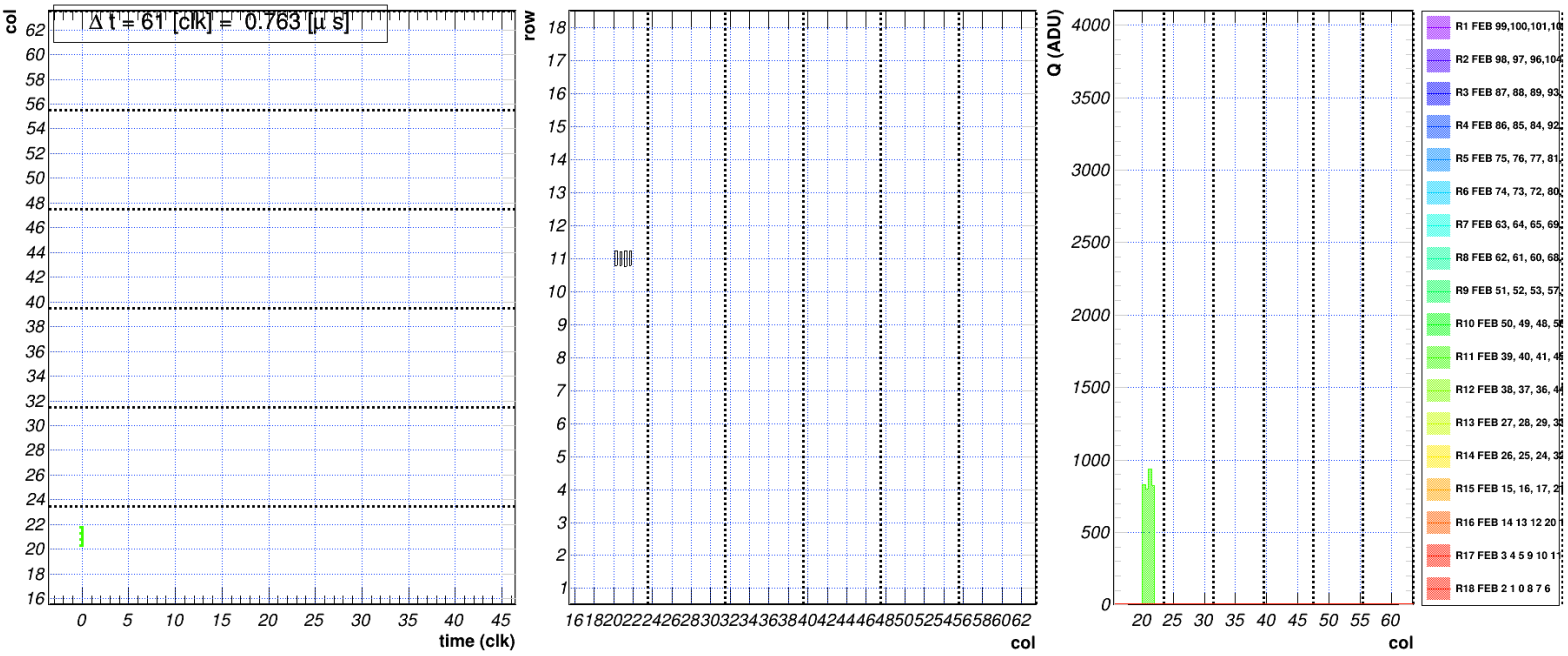


Figure 2.5: Clusterization of hits in TRD-2D [1]

are represented while on the horizontal axis we have the time (in clocks). Notice that there are four signals between columns 20 and 22 of the detector, at time zero. Also, I want to point out that in the legend on top of the plot, the time passed since the last interaction is noted (1.763 micro-seconds). The second plot shows the response of the readout channels, by indicating the column and row of the pad in the detector - and we see four vertical strips between columns 20 and 22 on row 11. The energy of the particles measured in each channel, is plotted in the third image, to the right.

Analyzing the three plots, we can say that the four signals were all measured at the same time, in consecutive channels and that their energy is coming from a single particle, hence we have a cluster of hits from which the original hit can be calculated.

2.1.3 Steering framework - inputs, tasks, macros, scripts

The FairRoot framework which represents the basis of cbmRoot is highly flexible; for instance it allows the user to run simulations with different detector configurations without having to recompile the code. This flexibility is given by the fact that in FairRoot, "everything is a task", a trait inherited by cbmRoot.
"A task is a procedure to read an already existing data level and produce a new one, or change the existing one" [1]. It is possible to create complex structures of tasks and subtasks, like the structure illustrated in Figure 2.6.
The basic functions of a task are:

- SetParContainer() - in which the parameter container is set, necessary for the execution of the task (*optional*);
- Init() - runs all initial steps like initializing variables, getting the parameters from the parameter container and getting the pointers to needed classes;

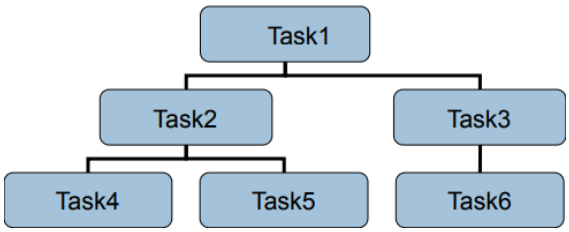


Figure 2.6: cbmRoot tasks structure example [1].

- Exec() - this is the method in which the main actions are run;
- Finish() - the method is called at the end of every run; all cleanup actions can and should be added here;
- ReInit() - in case the conditions of the run change, we can reinitialize the task(*optional*);
- FinishEvent() - all cleanup code goes here, that has to be run once the event was processed and the result was written to the output file (*optional*).

Once the task has been defined, it can only be run from a so-called steering macro. In this steering macro, the input and output files, as well as the parameter file have to be defined. Having done that, the RunManager object is initialized next. This object is used to run all the tasks needed (an illustration of this, is presented in Figures 2.7, 2.8, 2.9, 2.10).

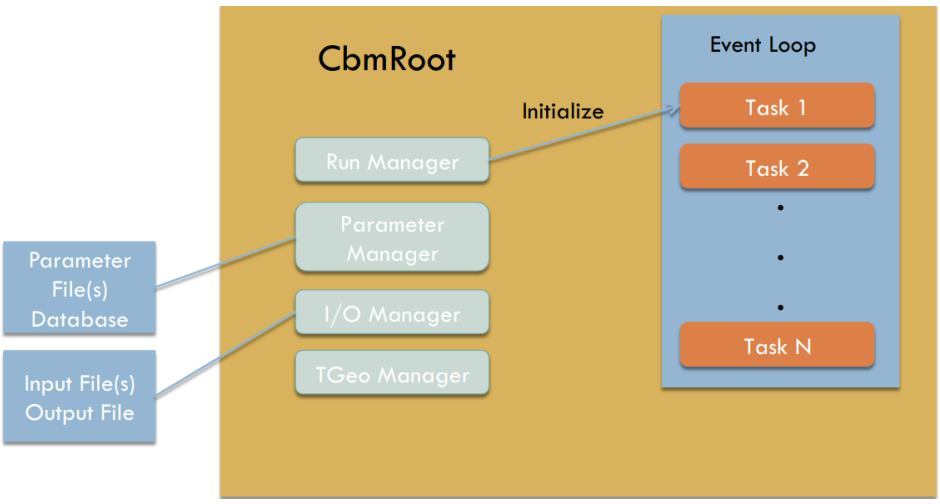


Figure 2.7: Call initialize task [1].

This architecture of tasks, macros and input files is what makes cbmRoot a very configurable and extendable framework, in which every developer can define her/his own task and add it to a steering macro, to be run for the entire collaboration. Also, tasks can be removed easily from those steering macros and thus each member of the collaboration can obtain the results she/he is interested in. Hence, the end goal of every developer for the CBM collaboration, is to create and test algorithms and eventually add them to the project as a task.
In order to test the code which will be later added as a new task in cbmRoot, simple scripts (also called macros - non-steering) can be written and tested by running them from the command line, using the following command:

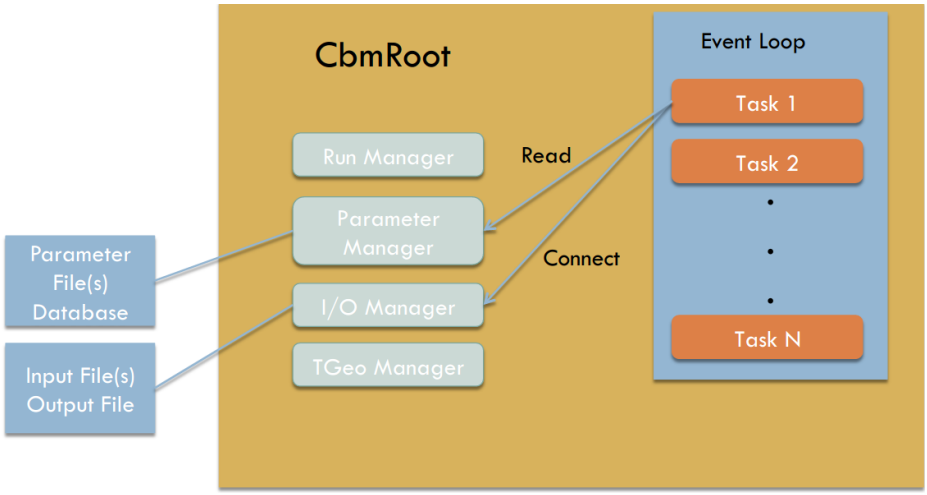


Figure 2.8: Task read parameters & connect to I/O [1].

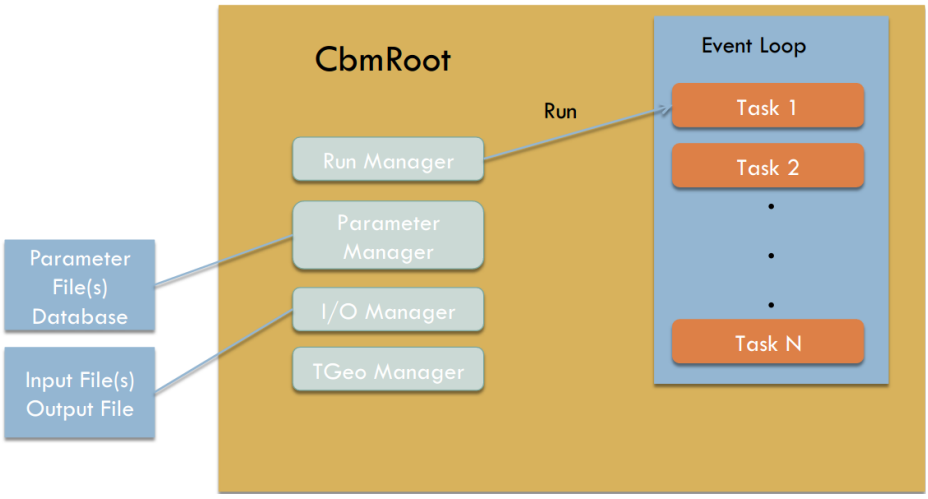


Figure 2.9: Run task [1].

root -l <macroName> (<parameters>)

Since the convention is to have macros which address only one main issue (one process), multiple macros would need to be run, in order to complete more than one process; for instance, in order to produce a digi file in simulations, we need to run two macros (mcbm_transport and mcbm_digi). Also, because simulations need to be run using different parameter files, setups or frequencies and we would need to produce many results (to have good statistics), we can add an extra level of automation by writing bash scripts in which macros are run for different configurations.

The code in Listing 2.1 shows an example of bash script, which walks through a list of simulation input files generated using the URQMD model (1000 files of Au-Au central collision at 10GeV) and for each one of those files, runs the transport macro and time based digitisation for two frequencies (10MHz and 100kHz). If we were to make a simple calculation, running the simulations manually would mean 1000 (no. of files) * 3 (number of macros called - 1 transport and 2 digitisation) = 3000 calls to add manually in the command line + creation of different folders for each run. With the bash script however, there is only one call to add manually which is a considerable automation of these processes. Additionally, bash scripts were essential in my

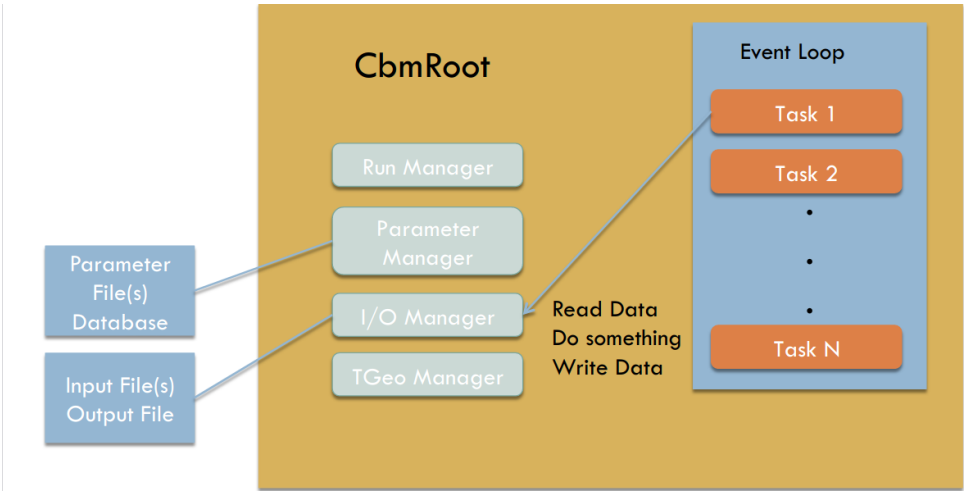


Figure 2.10: Write results to output [1].

work, given the fact that the simulations were run on the computer grid at the Nuclear Interactions and Hadronic Matter Centre of Excellence (NIHAM) of the HPD at NIPNE, in order to parallelise processes. The cbmRoot framework contains steering and non-steering macros for processing both data and simulations and most of the macros were copied and modified for the different mCBM runs and setups.

Listing 2.1 Example bash script for running multiple macros.

```
#! /usr/bin/bash
# Takes two parameters
# $1 - folder where input files are located
# $2 - folder where to place simulations
# $3 - file names (2021_07_survey or part or ....)
# Example usage: ". simulationsIncludingTransport.sh ~/Work/data/ ~/
   data/BatchSimulations 2021_07_survey"

function createDir()
{
    mkdir "$1"
}

function runTransportMacro() {
    root -l -q '$VMCWORKDIR/macro/mcbm/mcbm_transport.C('$2',"
       mcbm_beam_2021_07_surveyed","'$3'/'$4'","'$1'")'
}

function runDigitizationTimebased_10MHz {
    root -l -q '$VMCWORKDIR/macro/mcbm/mcbm_digi.C('$1',"'$2'/'$3'",1.
       e7,1.e7,0)'
}
```

```
function runDigitizationTimebased_100kHz {
    root -l -q '$VMCWORKDIR/macro/mcbm/mcbm_digi.C('$1','"'$2'/'$3'"',1.
       e5,1.e7,0)'
}
#####################################

NUMBEROFEVENTS=1000 # adjustable
PATH_TO_SIMULATIONFOLDER="$2"
OUTPUTFILENAMES="$3"

declare -a frequencies=("10MHz", "100kHz")
FILES=$(ls $1)
 for FILE in $FILES
     do
        FOLDERNAME="simulationsWithUrqmd_$(cut␣-d.␣-f5␣<<<␣$FILE)"
        createDir "$PATH_TO_SIMULATIONFOLDER/$FOLDERNAME"
        #creates "$HOME/data/BatchSimulations/
           simulationsWithUrqmd_000000" for instance
        for i in "${frequencies[@]}"
            do
             createDir "$PATH_TO_SIMULATIONFOLDER/$FOLDERNAME/$i"
             #creates "$HOME/data/BatchSimulations/
                simulationsWithUrqmd_000000/10MHz" for instance
             runTransportMacro "$1$FILE" "$NUMBEROFEVENTS" "
                $PATH_TO_SIMULATIONFOLDER/$FOLDERNAME/$i" "
                $OUTPUTFILENAMES"
             runDigitizationTimebased_$i "$NUMBEROFEVENTS" "
                $PATH_TO_SIMULATIONFOLDER/$FOLDERNAME/$i" "
                $OUTPUTFILENAMES"
           done
 done
```

2.2 Graphical representation of physical results

In High Energy Physics (HEP) experiments, the data which needs to be processed and visualised is so big, that simple graphs are not able to handle the amount of points that need to be added and would in any case result in images hard to read and assess. Instead, histograms are used, which are *"approximate representations of the distribution of numerical data"* [18]. In order to create the histograms, the entire data range is divided into intervals (called bins) and each interval then presents an estimation of data density.
The ROOT framework offers the user all tools needed for generating and working with histograms including re-binning, fitting or normalizing. Listing 2.2 presents the code, written in the root command line, to create a simple histogram of the Gaussian distribution of $2 * 10^4$ random numbers, while the result can be seen in Figure 2.11.

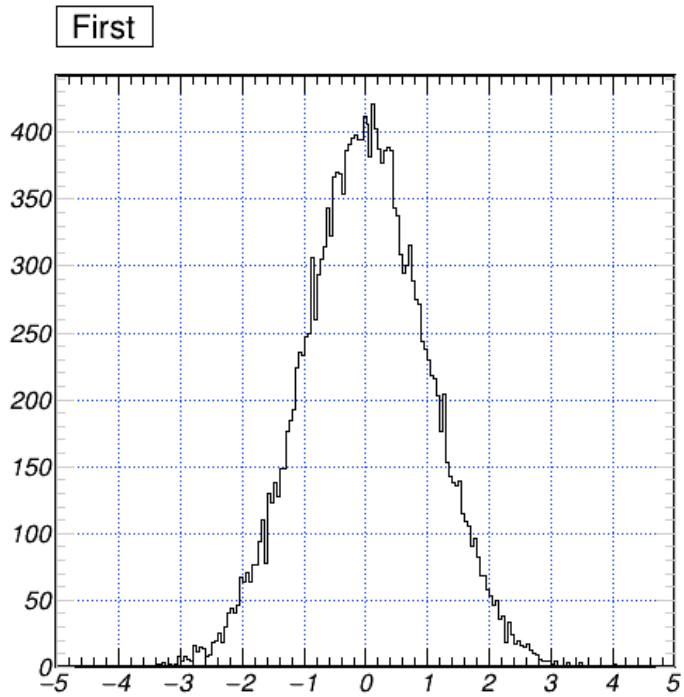


Figure 2.11: Gaussian distribution of random numbers.

Listing 2.2 Creating histogram - Gaussian distribution.

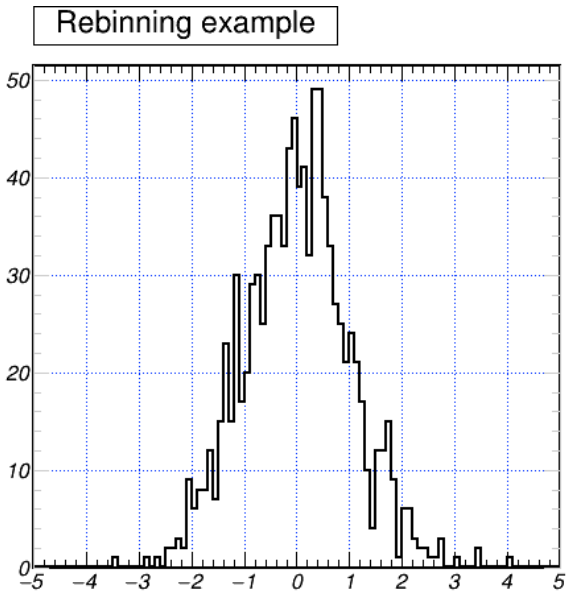
```
TH1F h1("h1","First",200,-5,5);
h1.FillRandom("gaus", 20000);
h1.Draw()
```

The class which is used for the creation of histograms is TH1F, which creates a 1-dimensional histogram with data of type float and for which there are 6 constructors available. The one used in this example, mentions the name of the object we want to create (h1), the title, the number of bins (200), and the limits of the x axis. The created object is then filled with $2 * 10^4$ random values in a Gaussian distribution. The Draw command outputs the result on screen.

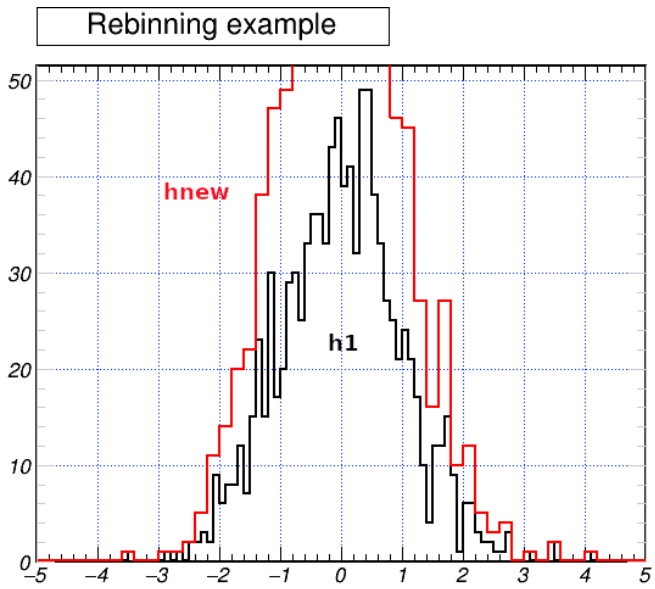
When creating a histogram, the entire range of values is divided into intervals (called bins) which are usually consecutive and contain the number of values that fall in that interval. The bins can be defined and modified in order to get a clearer image of the distribution of values. An example of how we can rebin a histogram, is presented in Listing 2.3 and figure 2.12. We've started out from 100 bins and are halving them, so in the *hnew* histogram there are only 50 bins to accommodate the data.

Listing 2.3 Rebinning a histogram

```
TH1F h1("h1","Rebinning␣example",100,-5,5);
h1.FillRandom("gaus", 1000);
TH1F *hnew=dynamic_cast<TH1F*>(h1.Rebin(2,"hnew"));
h1.Draw()
hnew->Draw("same")
```



(a) Initial histogram - 100 bins.



(b) Result of rebinning with both histograms drawn on canvas.

Figure 2.12: Rebinning a histogram

A very important operation when working with histograms in HEP is the "fitting" operation. *"Fitting is the method for modeling the expected distribution of events in a physics data analysis."*[18].There are two ways of doing this in ROOT - either by using the Fit panel of the Graphical User Interface (GUI), or programatically by using predefined or user defined functions. Figure 2.13 shows the Gaussian Fit created through the Fit Panel, for the initially created histogram.

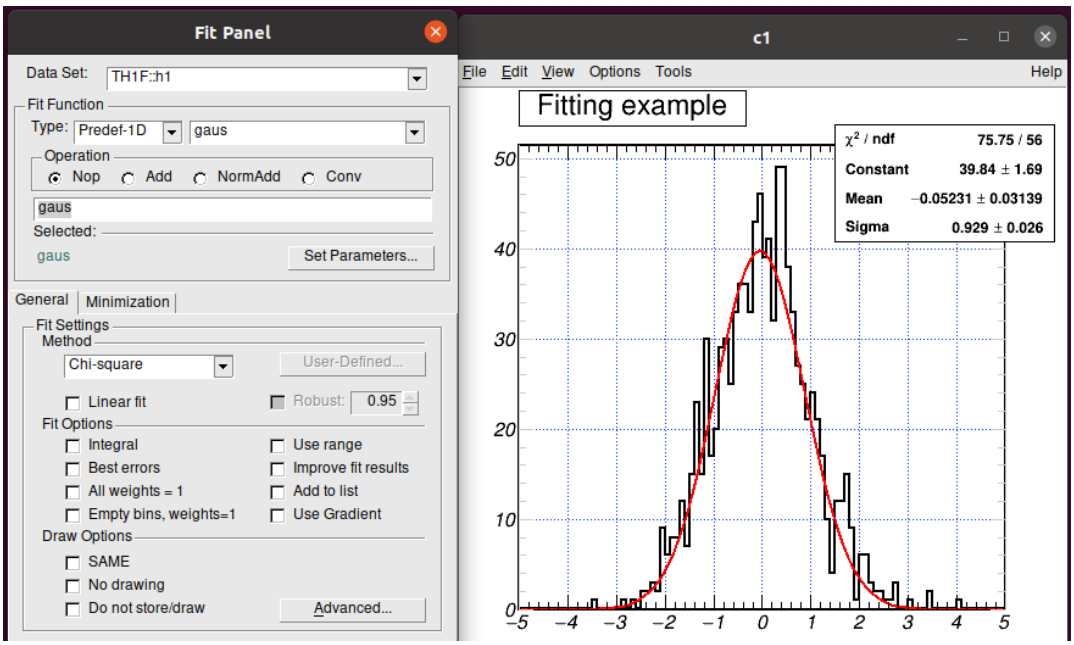


Figure 2.13: Fitting distribution on a given histogram.

The ROOT framework is also able to create 2D and 3D plots, both very useful tools in the visualisation of data.

Data in root files is organized in levels (an unlimited number) containing directories and objects

and is machine independent. The structure most commonly used is the TTree. Files resulted from cbmRoot simulations, are using the 'cbmsim' TTree structure to list the data for different detectors. The example in Figure 2.14 shows the cbmsim created after running the event reconstruction macro.

Every TTree structure contains branches and leafs and the data in the structure can be accessed by its path. Double clicking on a leaf in this tree, will result in a simple 1D histogram being created.

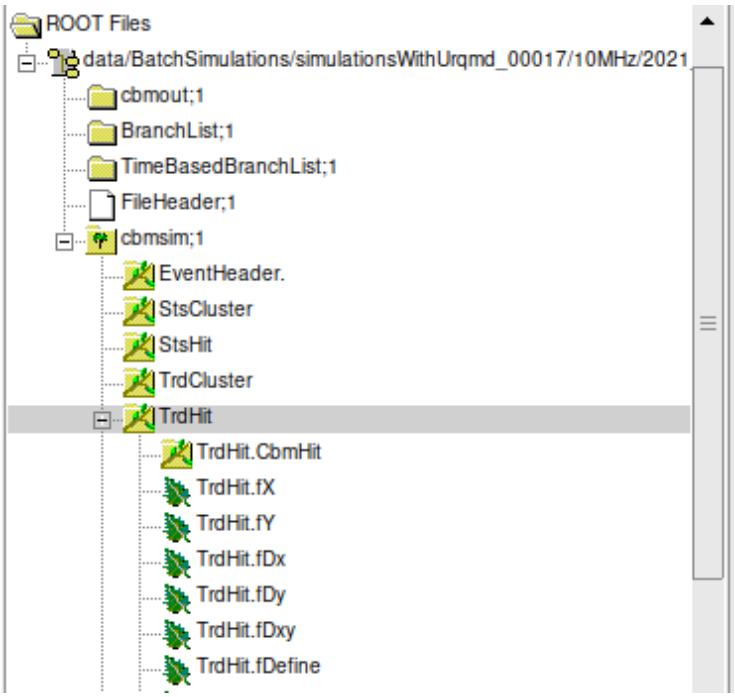


Figure 2.14: cbmsim structure from event reconstruction macro

In order to create 2D or 3D plots, one would need to use so-called "cuts"; for instance: **t.Draw("y:x")** will result in a 2D histogram of y as compared to x. A cut may also provide a condition for the data to be plotted; suppose x is a value in a range from -20 to 18, then **t.Draw("x", "x>1 && x<10")** will draw only those values of x between 1 and 10.
The example in Listing 2.4 shows how the 2D histogram represented in Figure 2.15 is created from the cbmsim tree in Figure 2.14, displaying the position (x:y) of the simulated hits in the TRD-2D.

Listing 2.4 Creating 2D histogram - simulated hit position in TRD-2D.

```
cbmsim->Draw("TrdHit.fY:TrdHit.fX","TrdHit.fAddress==5","colZ");
```

The code for the simulation of the behaviour of the TRD-2D detector has been implemented considering it as a part of the TRD subgroup of detectors; hence, in order to isolate this detector and exclude all hits to the rest of the TRD detectors, in the cut address 5 has to be specified. The hybrid TRD-2D detector is to be found at address 21 while for TRD-1D, either address 37 ore 53 are in use.
Notice that because cbmsim is the root of the structure, it can be addressed directly.

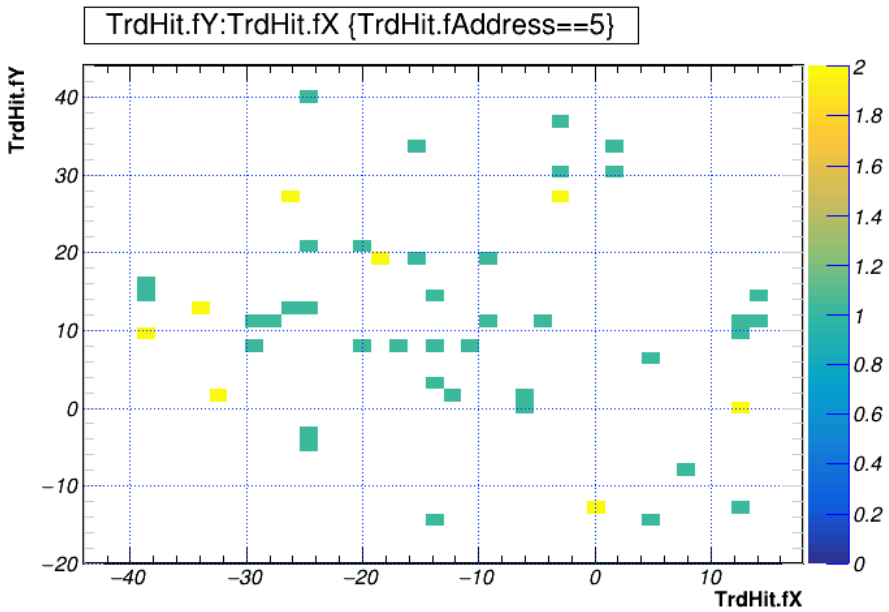


Figure 2.15: 2D plot - Simulated hit position in TRD-2D

2.3 Online/offline event selection

Not all the data pushed by the detectors in CBM to the data acquisition system is important to the physics case proposed by the experiment and since the detectors will be spewing out some petabytes (PB) of data at every run, it is not feasible to store all this information. That is why, an online event selection has to be made from this data. An offline event selection is the process in which the algorithms for the online event selection are created and/or optimized. The offline event selection looks at data taken in test runs (which has not been pre-processed) and at simulations using Monte-Carlo models. In both cases, all data is analysed (minimum-bias). The selection is made using algorithms which are programmed into tasks in order to have reusable code which will be able to perform in the same way both online and offline.

2.3.1 Big data is too big to be confined

The CBM is an experiment in which heavy nuclei are collided at rates of up to 10MHz in order to obtain the high statistics needed for the measurement of rare probes. In addition to this, no triggering system will be used, meaning that the front end electronics (FEE) of each detector in the experiment, will be sending data to the data acquisition system freely. The raw data rate that is expected at the CBM is of 1TB/s. The data is transported from the data acquisition system (DAQ) to the GreenCube data center using InfiniBand connections which provide very low latency and high throughput, so the transport of data is not an issue. Storing the data however, is a real challenge since the cost of storage is quite high. This is why, the online event selection is implemented in the DAQ where a First-Level-Event-Selection cluster processes the data - building timeslices from all readings received, then reconstructing tracks and events. The FLES is a heterogeneous system which uses FPGAs and GPUs in order to achieve the processing power needed for the online selection task (which is of 1TB of data every second).

Big data is indeed too big to confine and Figures 2.16 and 2.17 were chosen in order to

illustrate the complexity of the event selection task and the massive rate at which data is sent to the FLES. These images show the TRD-2Ds response during mCBM run number 2365, in two consequent timeslices - run 2365 was a relatively low interaction run, using oxygen on nichel collisions.

The same correlations can be performed here, as for Figure 2.4 (hits are correlated in time, position and energy). Notice that the time between these two timeslices is of 0.263 microseconds and notice the amount of particles which pass through the detector and are read out in that time. The lower frame of the images shows all the hits detected in that timeslice. In the central frame, the small plots with a Gaussian distribution indicate the energy of the particle, while the top frame represents the time of interaction.

2.3.2 The different levels to event selection

In other hadronic physics experiments, the acquisition of data is done using the triggered model and that means that the event selection is done by that one reference detector.
At CBM however, the event selection must be done after the data is read out. In order to reconstruct an event, starting from hit clusters in timeslices, there are three different approaches or levels, each one adding to the precision of the reconstruction. In a first instance, one can try to mimic in software the event selection done by systems that use triggered data acquisition. This can be done by choosing a reference detector and comparing the data from other detectors to the data recorded by that one reference detector. This however, is an extremely imprecise approach in the case of CBM - a visual explanation to why this is not suitable for free-streaming experiments, can be found in Chapter 2.4.1.
An additional level to event reconstruction is the definition of clear rules for detector hits, in order to define an event. For instance, an event will be defined as time correlated hits, with at least one hit in STS, 2 in the TRD-2D and 4 hits in ToF. If in any timeslice, there are such correlations to be found, then it can be concluded that there is an event.
The last level of event selection, the one that helps reconstruct the event with the highest degree of precision, is event reconstruction by using tracking. Tracks are reconstructed using Cellular Automaton (CA) and the Kalman filter [2] and the algorithms for tracking are currently being optimized and extended to accommodate all detectors present in the experiment.

This last approach to event selection is used in the FLES and this is why, besides the massive computing power that is obtained by using GPUs and FPGAs, an optimization of the algorithms is needed in order to make use of the parallel computing capabilities of the hardware.

2.4 Experimental results

2.4.1 Motivation of trigger-less data acquisition

The difference between triggered and trigger-less data acquisition has already been discussed in previous sections. Also, it is clear that the CBM is an experiment which uses high interaction rates in order to generate as much matter as possible, a basic condition to finding rare probes.

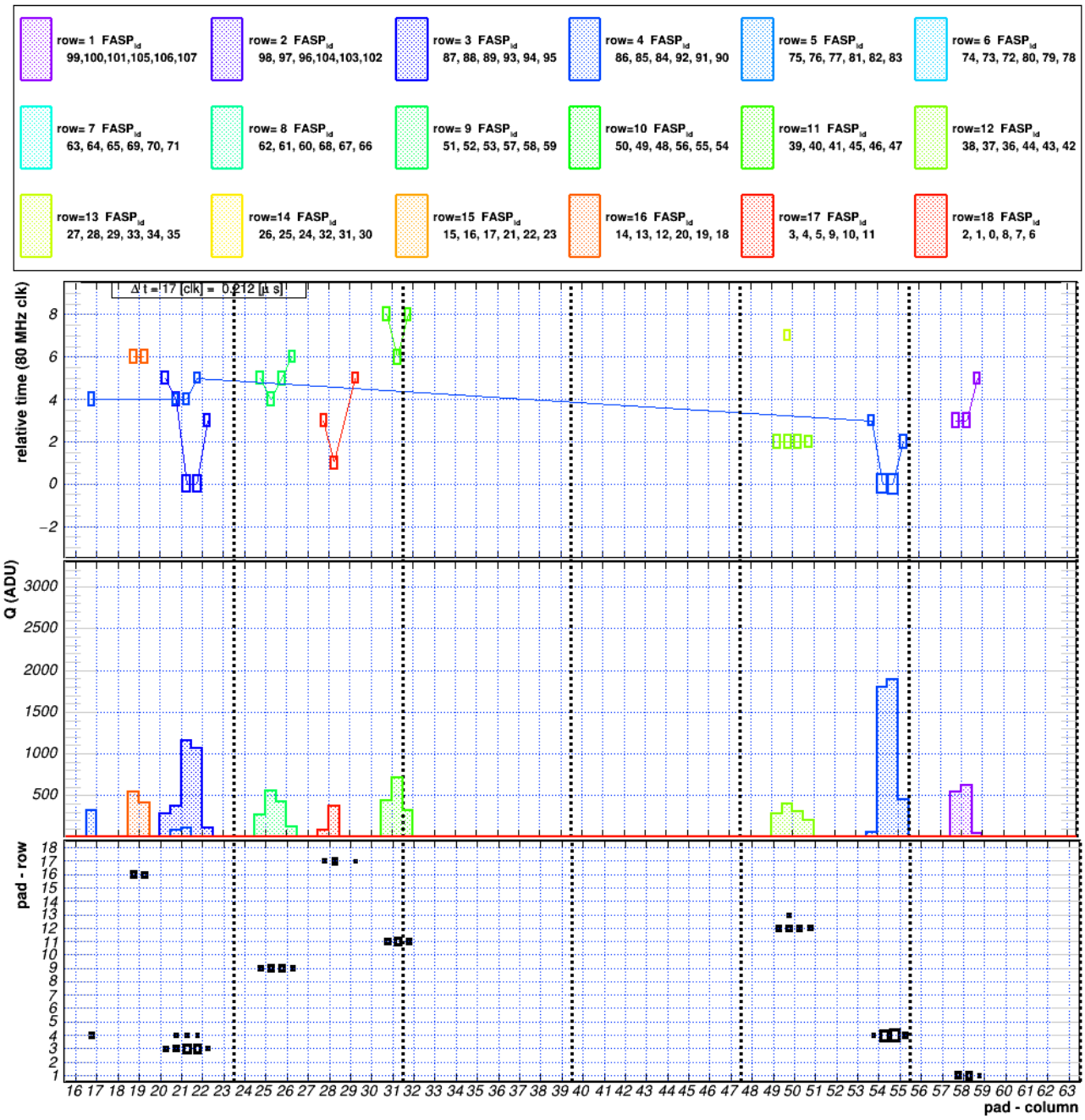


Figure 2.16: TRD-2D hits in mCBM run 2365 - timeslice 1. [2]

In addition to this, a very high multiplicity is expected, meaning that one collision should be able to create a big number of particles and tracks. In order to create high multiplicities, heavy nuclei will be used in the collisions. Figures 2.18 and 2.19 represent two plots which I have generated in order to motivate why the triggered acquisition model would not be proper in the CBM experiment. The two images are obtained in simulations and show the difference between a heavy system (collision of gold on gold) and a light system (collision of oxygen and nichel). The trait that makes these systems heavy or light is the number of nucleons they contain (protons and neutrons). Oxygen contains 8 protons and 8 neutrons, nichel - 28 protons and 34 neutrons and gold - 79 protons and 118 neutrons. Hence, a collision of two gold nuclei will produce a higher multiplicity than a collision of oxygen on nichel.

[2] Courtesy of Dr. Alexandru Bercuci - HPD, NIPNE, Măgurele, Romania

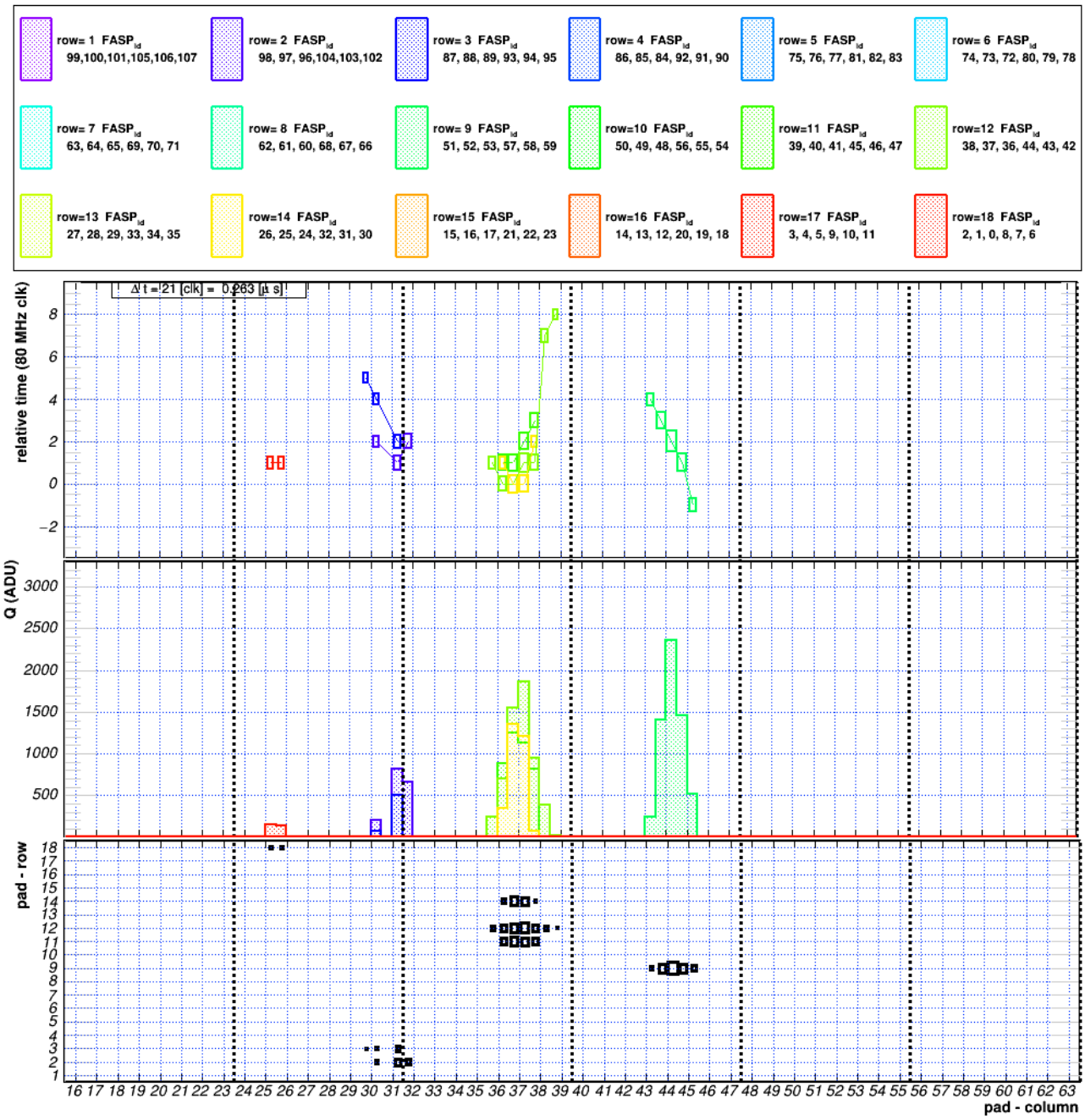


Figure 2.17: TRD-2D hits in mCBM run 2365 - timeslice 2.

I have investigated the hits generated at the same interaction rate of 10MHz, by the two types of systems and simulated the triggered acquisition system by taking as reference the Monte-Carlo points and tracks which have generated the hits. After running the existing transport, timebased digitisation and event reconstruction macros already existing in cbmRoot, I have designed and coded a macro to correlate the hits for TRD. The macro takes as input, the tra.root, digi.root and rec.root files generated by the previous steps - these files have a TTree structure, so the macro creates references only to the branches needed (see Listing 2.5).

Listing 2.5 Input for Hit correlation macro.

```
if(!TFile::Open(Form("%s.rec.root", fn))) return;
```

```
TTree *trec(nullptr);
if(!(trec = (TTree*)gFile->Get("cbmsim"))) return;
CbmEvent *ev(nullptr);
TClonesArray *evs(nullptr);
FairEventHeader *evhead(nullptr);
TClonesArray *hitsSts(nullptr);
TClonesArray *clsTrd(nullptr), *hitsTrd(nullptr);
TClonesArray *hitsTof(nullptr);
trec->SetBranchAddress("CbmEvent", &evs);
trec->SetBranchAddress("EventHeader.", &evhead);
trec->SetBranchAddress("StsHit", &hitsSts);
trec->SetBranchAddress("TrdHit", &hitsTrd);
trec->SetBranchAddress("TrdCluster", &clsTrd);
trec->SetBranchAddress("TofHit", &hitsTof);

if(!TFile::Open(Form("%s.digi.root", fn))) return;
TTree *tdig(nullptr);
if(!(tdig = (TTree*)gFile->Get("cbmsim"))) return;
vector<CbmTrdDigi> *digi(nullptr);
vector<CbmMatch> *trdMatch(nullptr);
CbmMCEventList *mcevl(nullptr);
CbmTimeSlice *timeslice(nullptr);
tdig->SetBranchAddress("TrdDigi", &digi);
tdig->SetBranchAddress("TrdDigiMatch", &trdMatch);
tdig->SetBranchAddress("MCEventList.", &mcevl);
tdig->SetBranchAddress("TimeSlice.", &timeslice);

if(!TFile::Open(Form("%s.tra.root", fn))) return;
TTree *tmc(nullptr);
if(!(tmc = (TTree*)gFile->Get("cbmsim"))) return;
TClonesArray *trdPoint(nullptr);
tmc->SetBranchAddress("TrdPoint", &trdPoint);
TClonesArray *mcTracks(nullptr);
tmc->SetBranchAddress("MCTrack", &mcTracks);

TFile *fout = TFile::Open("hitCorrel.root", "RECREATE");
```

From the transport macro we get the Monte-Carlo generated points and tracks, from the digitisation macro we get the timeslices, values of TRD hits and the Monte-Carlo events list, while from the reconstructed events we get the event information and the TrdHits together with their clusters. Then the output is defined - a root file called hitCorrel, which creates a TTree containing : the TRD address (one of the three: 5 - TRD-2D, 37 and 53 - TRD-1D), the position of the hit, it's energy, particle id (there is a mapping between ids and types of particles and this id is deduced based on the energy), the mother (particles which come from the primary interaction have a -1 in this field), timeslice id and the id of the reconstructed event (see Listing 2.6).

Listing 2.6 Output definition for Hit correlation macro.

```
TFile *fout = TFile::Open("hitCorrel.root", "RECREATE");
TTree *tout = new TTree("thc", "Hit correlation in the mCBM21");
tout->Branch("address", &add, "address/I");
tout->Branch("xTrd", &xTrd, "xTrd/F");
tout->Branch("yTrd", &yTrd, "yTrd/F");
tout->Branch("tTrd", &tTrd, "tTrd/F");
tout->Branch("ETrd", &eTrd, "ETrd/F");
tout->Branch("n0", &n0, "n0/I");
tout->Branch("ev", jev, "ev[n0]/I");
tout->Branch("x0", xp0, "xp0[n0]/F");
tout->Branch("y0", yp0, "yp0[n0]/F");
tout->Branch("t0", tp0, "tp0[n0]/F");
tout->Branch("E0", Ep0, "Ep0[n0]/F");
tout->Branch("pid", partId, "partId[n0]/I");
tout->Branch("mother", pmother, "pmother[n0]/I");
tout->Branch("evRecId", &evid, "evRecId/I");
tout->Branch("timesliceId", &timesliceId, "timesliceid/I");
```

For each reconstructed event, we compute the TRD hit that is part of that event and for that hit we get all the other hits that make up its cluster. For each of these hits, we extract the Monte-Carlo point that is behind it and the time of the point as compared to the beginning of the event - see (remember that we know this because we're in simulations and using the Monte-Carlo event as reference). The code for these calculations is in Listing 2.7.

Listing 2.7 Hit correlation macro.

```
  for(int iev(0); iev<evs->GetEntriesFast(); iev++) { // get all
     reconstructed events
    if(!(ev = (CbmEvent*)(*evs)[iev])) continue;
    evid = ev->GetNumber();
    if(VERBOSE>1) printf("Event number: %d \n", evid);
    ntrd      = max(0, ev->GetNofData(ECbmDataType::kTrdHit)); // get
        all hits in event

    for(Int_t itrd(0); itrd<ntrd; itrd++, nTrdTot++) {
      auto hTrdId = ev->GetIndex(ECbmDataType::kTrdHit, itrd); //
         index of trd hit
      hTrd = (CbmTrdHit*)(*hitsTrd)[hTrdId];
      if(VERBOSE>1) printf("TRD Hit Addr: %d \n", hTrd->GetAddress()
         );
      if(VERBOSE>1) printf("TRD Hit Index: %d \n", hTrdId);

      add = hTrd->GetAddress();
      xTrd = hTrd->GetX();
```

```
        yTrd = hTrd->GetY();
        tTrd = hTrd->GetTime() + timesliceId*timesliceLength;
        eTrd = hTrd->GetELoss();

        Int_t ClusterIndex = hTrd->GetRefId(); // Index of the cluster
            where hit was registered
        if (VERBOSE>1) printf("ClusterIndex:␣%d\n", ClusterIndex);
        if(!(cTrd = (CbmTrdCluster*)(*clsTrd)[ClusterIndex])) {
          continue;
        }

        n0 = 0;
        for(auto id : cTrd->GetDigis()) { // get all hits in the
            cluster
          CbmTrdDigi d = (*digi)[id];
          CbmMatch dm = trdMatch->at(id);

          int row = d.GetAddressChannel()/72,
              col = d.GetAddressChannel()%72,
              asc = (col/8) - 5;

          vector<CbmLink> links = dm.GetLinks();

          for (auto it : links) {
            int pindex = it.GetIndex(); // index of the point from MC
            mcevlEventTime = mcevl->GetEventTime(it.GetEntry(), 0); //
                get Monte Carlo event time
            CbmTrdPoint *p = (CbmTrdPoint*)((*pointsMap)[pindex]); //
               get TRD point

            // get point time depending of MC event start time
            float pointTime = p->GetTime() + mcevlEventTime;

            xp0[n0] = (p->GetXIn() + p->GetXOut()) / 2;
            yp0[n0] = (p->GetYIn() + p->GetYOut()) / 2;
            tp0[n0] = pointTime;
            Ep0[n0] = p->GetEnergyLoss();

            auto tracks = mcTracksMap[lastEventInMemory];
            CbmMCTrack *track = (CbmMCTrack*)((*tracks)[p->GetTrackID
               ()]);
            partId[n0]  = track->GetPdgCode();
            pmother[n0] = track->GetMotherId();

            n0++;
          }
        }
```

```
        tout->Fill();
      }
    }
```

With the root file resulted from this macro, I then further continued the analysis, designing and implementing another macro with the main task of plotting the results in order to have a graphical representation. The code of this macro is fairly simple and uses standard ROOT directives for drawing histograms (See Listing 2.8).

Listing 2.8 Drawing the hit correlations.

```
TFile *f = TFile::Open(fn);
if(!f) return;

TTree *thc = (TTree*)f->Get("thc");
if (!thc) {printf("No thc tree available!\n"); return;}
gStyle->SetPalette(1);

TCanvas *c1 = new TCanvas("c1","c1");
TLegend *leg = new TLegend();

// Draw time correlations
thc->Draw("t0:tTrd", "mother==-1");
thc->Draw("t0:tTrd", "mother!=-1", "same");

TGraph *trd = (TGraph*)c1->GetListOfPrimitives()->At(3);
trd->SetMarkerStyle(7);
trd->SetMarkerColor(2);
trd->SetName("Primary particles");
leg->AddEntry(trd, trd->GetName(), "p");

TGraph *trd2 = (TGraph*)c1->GetListOfPrimitives()->At(4);
trd2->SetMarkerStyle(7);
trd2->SetMarkerColor(1);
trd2->SetName("Secondary particles");
leg->AddEntry(trd2, trd2->GetName(), "p");

TH2F *h = (TH2F*)c1->GetListOfPrimitives()->At(1);
TString hTitle = "Plot Hit Correlations";
h->SetTitle(hTitle);
leg->Draw();

c1->Modified();

strcpy(saveToFile, foldername); // copy string one into the result.
strcat(saveToFile,"/hitCorrelationPlot_time");
```

```
c1->SaveAs(saveToFile);
```

What is clearly visible in the images is that a light system (one in which the collisions are between elements that contain only few protons and neutrons, thus generating a small multiplicity) can use the triggered acquisition model because the event reconstruction takes into account almost all hits from the other detectors and not much data is lost in the process. In a heavy system with high multiplicity however, using a triggered acquisition model would mean discarding a big amount of data and that is less than desirable when trying to study rare probes. If we were to extrapolate on this finding, the heavier the system, the less accurate the event selection would be.

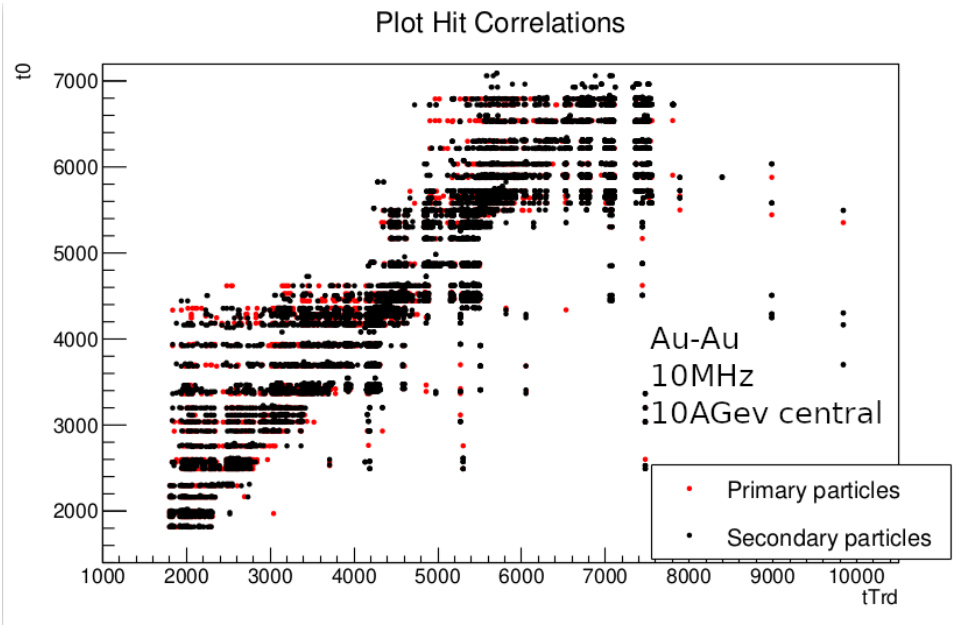


Figure 2.18: Correlation of hits in heavy systems.

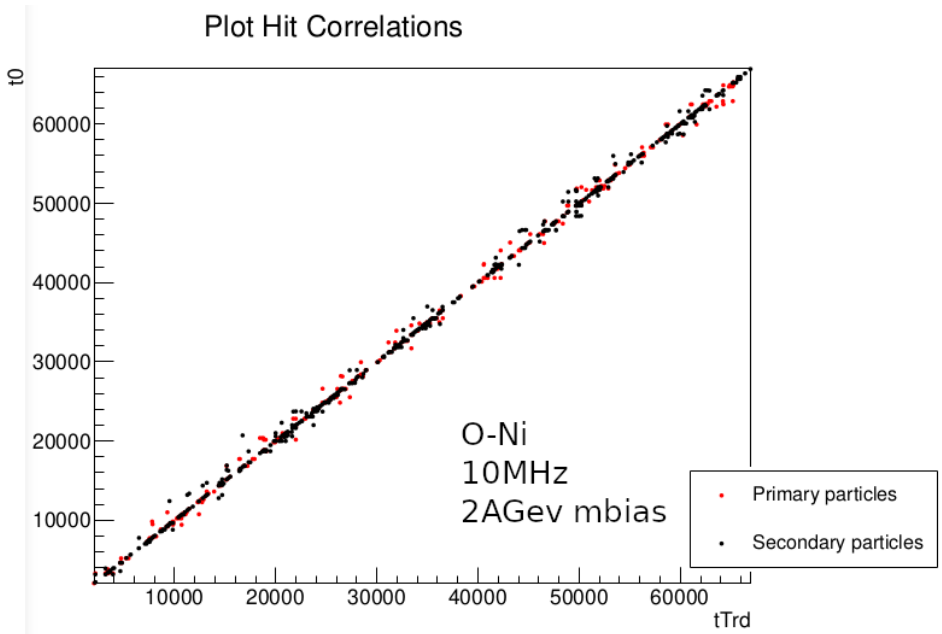


Figure 2.19: Correlation of hits in light systems.

2.4.2 Data time correlations and event definition

The time correlation check between real data and simulations is an important step in vakidating and checking the accuracy of the simulations. There is a macro in cbmRoot - $check_timing_any$, which checks for time correlations between two detectors. The time correlation compares the timing of the hits which make up each offline reconstructed event. This macro was run on both simulations and on data, using the same setup in the simulations as the ones from the data taking run. The resulting root files then constituted the input to my macro (see Listing 2.9) which created a histogram for comparison.

Listing 2.9 Drawing time correlations between data and simulations.

```
void drawTimeCorrelationsBetweenDataAndSimulations(const Char_t *dfn="
   ~/data/1588.tck.root", const Char_t *sfn="~/data/
   resultsInputAna_alice/limited/0/100kHz/HistosTimeCheck_001.root",
   const Char_t *foldername="~/data/") // data file name (1588.tck.
   root), simulation file name (HistosTimecheck_001.root) and folder
   name for saving output
{
    gStyle->SetOptDate(0);
   const char* const leafNames[1] = {"hTrdTofDiff0"};

    TFile *f = TFile::Open(dfn);
    if(!f) return;

    TFile *simf = TFile::Open(sfn);
    if (!simf) return;

    int idx(0);
    for (auto leaf:leafNames) {
        TCanvas *c1 = new TCanvas(Form("c%d", idx),"c1");
        c1->SetLogy();
        TDirectoryFile* tt = (TDirectoryFile*)f->Get("Trd");
        TH1D* hTrdTofDiff = (TH1D*)tt->Get(leaf);
        TF1* ff = hTrdTofDiff->GetFunction("gs_trd");
        hTrdTofDiff->SetDirectory(gROOT);

        hTrdTofDiff->RebinX(10);
        hTrdTofDiff->SetMarkerStyle(20);hTrdTofDiff->SetMarkerColor(
           kRed);
        hTrdTofDiff->Fit(ff);
        ff = hTrdTofDiff->GetFunction("gs_trd"); ff->SetLineColor(kRed
           );
        float offset = ff->GetParameter(3);
        hTrdTofDiff->Scale(1./offset);
        hTrdTofDiff->Fit(ff);
```

```
TDirectoryFile* strd = (TDirectoryFile*)simf->Get("Trd");
TH1D* hTrdTofDiffSim = (TH1D*)strd->Get(leaf);
TF1* fs = hTrdTofDiffSim->GetFunction("gs_trd");
hTrdTofDiffSim->SetName(Form("%sSim", hTrdTofDiffSim->GetName
   ()));
hTrdTofDiffSim->SetMarkerStyle(24);hTrdTofDiffSim->
   SetMarkerColor(kBlue);
hTrdTofDiffSim->SetDirectory(gROOT);

hTrdTofDiffSim->RebinX(10);
hTrdTofDiffSim->Fit(fs);
fs = hTrdTofDiffSim->GetFunction("gs_trd"); fs->SetLineColor(
   kBlue);
hTrdTofDiffSim->Fit(fs,"", "e1", -500, 30);
float offsetsim = fs->GetParameter(3);
hTrdTofDiffSim->Scale(1./offsetsim);
hTrdTofDiffSim->Fit(fs,"", "e1", -500, 30);
fs->SetRange(-500, 500);

fs->SetParNames("max.␣signal", "mean", "sigma", "noise");

gROOT->cd();
if (idx) hTrdTofDiffSim->GetXaxis()->SetRangeUser(-200, 200);
hTrdTofDiffSim->Draw("p");
hTrdTofDiff->Draw("p␣same");
hTrdTofDiffSim->SetTitle("TRD-2D␣/␣ToF␣time␣difference");
hTrdTofDiff->SetTitle("");

// change Y Axis title and position
hTrdTofDiffSim->GetYaxis()->SetTitle("Relative␣counts");
hTrdTofDiffSim->GetYaxis()->SetTitleOffset(1.3);

 // change X Axis title and position
hTrdTofDiffSim->GetXaxis()->SetTitle("Time␣difference␣[ns]");
hTrdTofDiffSim->GetXaxis()->SetTitleOffset(1.3);

TLegend *leg = new TLegend();

leg->SetNColumns(3);
leg->AddEntry((TObject*)nullptr, "", NULL);
leg->AddEntry((TObject*)nullptr, "Simulations", NULL);
leg->AddEntry((TObject*)nullptr, "Measurements", NULL);

leg->AddEntry((TObject*)nullptr, "DATA", NULL);
leg->AddEntry(hTrdTofDiffSim, "␣", "p");
leg->AddEntry(hTrdTofDiff, "␣", "p");
```

```
        leg->AddEntry((TObject*)nullptr, "Model", NULL);
        leg->AddEntry(fs, "␣", "l");
        leg->AddEntry(ff, "␣", "l");
        leg->AddEntry(hTrdTofDiff, "mCBM␣Run␣1588", "p");
        leg->AddEntry(ff, "S/N␣mCBM␣fit", "l");

        leg->AddEntry(hTrdTofDiffSim, "MC␣1588", "p");
        leg->AddEntry(fs, "S/N␣MC␣fit", "l");

        leg->Draw();
        c1->Modified();

        char saveToFile[100];    // array to hold the result.
        strcpy(saveToFile, foldername); // copy string one into the
           result.
        strcat(saveToFile,"/timeDiff_TRDAddr5"
       c1->SaveAs(saveToFile);
       c1->Close();
       idx++;
    }
}
```

In both histograms from Figure 2.20, the time difference between the TRD-2D and the ToF detector is represented - the histogram in red was created from real data while the one in blue is obtained in simulations. The Gaussian distribution around time zero means that most of the hits in the TRD-2D and the hits in ToF have arrived at approximately the same time and the time difference is negligible. However, looking at the simulations compared to the data, we see that in simulations there is a factor of 10 difference in S/N ratio and that a spurious component appears.
Thus, I have managed to show that the simulations do require optimization in order to take into account the configuration and response of the front-end electronics.

2.4.3 Quality assurance procedures

In the process of event selection the algorithms employed for reconstructing the tracks between hits use Cellular Automaton and the Kalman filter. In order to improve these algorithms and ensure that the tracks are not only correctly reconstructed but also optimal, the community needs so-called quality assurance procedures. These procedures use Monte-Carlo simulations to create the tracks and select events and compare the result to the one from offline event reconstruction.

2.4.3.1 Event reconstruction from simulations

The event reconstruction from simulations is a macro which I created in order to help with quality assurance. It has meanwhile been implemented as a task and included in the cbmRoot repository.

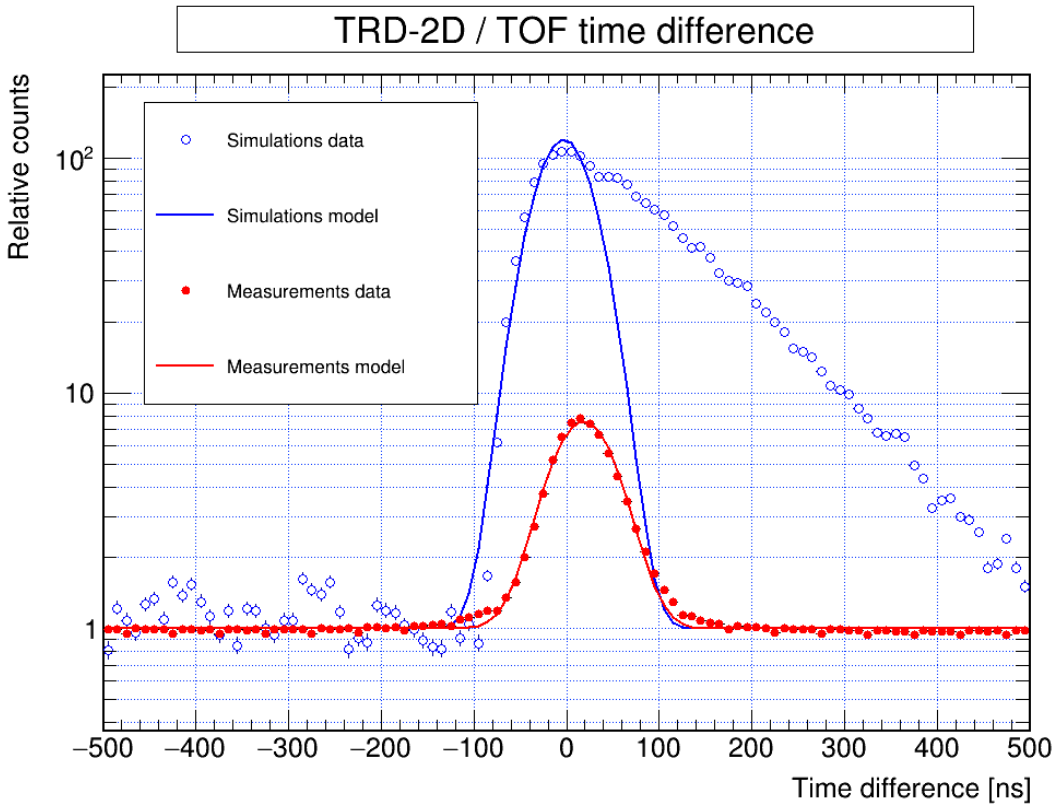


Figure 2.20: TRD-2D - ToF Time correlation - comparison between simulation and data.

The input to this macro is the same as the one for the macro in Listing 2.5, so the root files from event reconstruction, digitisation and transport. For each reconstructed event, we get the TRD hits that are part of that event; then for each hit we get its cluster and find the points that are part of the Monte-Carlo hit. We register the position, energy, time, particle id, track id and mother of the MC hit. This results in a root file which is then used in a drawing macro (Listing 2.10), to produce images of the reconstructed events. An example of drawing resulted from these macros is the one in Figure 2.21.

Listing 2.10 Draw match event reconstruction to MC points.

```
#include "/home/adriana/Work/macros/hitRecMc.h"

void drawMatchX( std::vector<hitRecMc> *trd, std::vector<hitRecMc> *
   sts, int eventId, const Char_t *saveFolder ) {
    int ih, ip, ihe;
    std::vector<TGraph*> graphsDrawn;
    std::vector<int> tracks;

    gStyle->SetPalette(1);

    TCanvas *c1 = new TCanvas(Form("c%d", eventId), Form("c_event%d",
       eventId));
    TLegend *leg = new TLegend();
    gStyle->SetOptDate(0);
```

```
gStyle->SetOptStat(0);
gStyle->SetOptFit(0);

TGraphErrors *gh (nullptr);
TGraph *gp (nullptr);

/*TRD HITS*/
for (auto hit : (*trd)) {
    hit.ToString();
    for (auto mcpoints: hit.MC) {
        cout<<"␣␣"; mcpoints.ToString();
        if (mcpoints.trackid == -1) continue;
        if (tracks.size() != 0) {
            std::vector<int>::iterator itf= std::find(tracks.begin
               (), tracks.end(),mcpoints.trackid);
            if (itf != tracks.end()) {
                continue;
            }
        }
        std::cout << "Found␣new␣trd␣track␣@␣" << mcpoints.trackid
           << "\n";
        tracks.push_back(mcpoints.trackid);
    }
}

for (auto hit : (*sts)) {
    hit.ToString();
    for (auto mcpoints: hit.MC) {
        cout<<"␣␣"; mcpoints.ToString();
        if (mcpoints.trackid == -1) continue;
        if (tracks.size() != 0) {
            std::vector<int>::iterator itf= std::find(tracks.begin
               (), tracks.end(),mcpoints.trackid);
            if (itf != tracks.end()) {
                continue;
            }
        }
        std::cout << "Found␣new␣track␣@␣" << mcpoints.trackid << "
           \n";
        tracks.push_back(mcpoints.trackid);
    }
}

bool firstIteration = true;
for (auto trkid : tracks) {
    cout << "drawing␣" << trkid << "\n";
```

```
        gh = new TGraphErrors(); gh->SetMarkerStyle(24); gh->
           SetMarkerSize(2);  ih = 0;
        gp = new TGraph(); gp->SetMarkerStyle(29); gp->SetMarkerSize
           (2); gp->SetMarkerColor(kRed); ip = 0;

        gh->SetName(Form("gh%d", trkid));
        gp->SetName(Form("gp%d", trkid));

        for (auto hit : (*trd)) {
            bool hregister = true;
            for (auto mcpoints: hit.MC) {
                if (mcpoints.trackid != trkid) continue;
                gp->SetPoint(ip++, mcpoints.z, mcpoints.x);
                if (hregister) {
                    gh->SetPoint(ih, hit.z, hit.x);
                    gh->SetPointError(ih, hit.dz, hit.dx);
                    ih++;
                    hregister = false;
                }
            }
        }

        for (auto hit : (*sts)) {
            bool hregister = true;
            for (auto mcpoints: hit.MC) {
                if (mcpoints.trackid != trkid) continue;
                gp->SetPoint(ip++, mcpoints.z, mcpoints.x);
                if (hregister) {
                    gh->SetPoint(ih, hit.z, hit.x);
                    gh->SetPointError(ih, hit.dz, hit.dx);
                    ih++;
                    hregister = false;
                }
            }
        }
        graphsDrawn.push_back(gh);
        graphsDrawn.push_back(gp);

        if (firstIteration) {
            leg->AddEntry(gp, "MC␣point", "p");
            leg->AddEntry(gh, "Reconstr.␣hit", "p");
            leg->AddEntry(gh, "Tracks", "l");
            firstIteration = false;
        }
    }

    TH1 *h = new TH2I("h",";z␣(cm);x␣(cm)",100,-10,220,100,-100,100);
```

```
    h->Draw("p");
    h->SetTitle(Form("Match␣MC␣Points␣to␣Event␣Reconstruction␣-␣event␣
       %d", eventId));
    int drawingOptions_linecolors[21] = {41, 40, 42, 43, 45, 46, 30,
       32, 34, 38, 20, 28, 29, 12, 5, 4, 3, 7, 9, 8, 6};
    int indexcolor = 0;
    for (auto g : graphsDrawn) {
        g->Fit("pol1");
        TF1 *f1 = g->GetFunction("pol1");
        if(f1) {
            f1->SetRange(0, 230);
            std::cout << "indexcolor:␣" << indexcolor << "␣-␣color␣is:
               ␣" << drawingOptions_linecolors[indexcolor] << "\n";
            if (drawingOptions_linecolors[indexcolor]) {
                f1->SetLineColor(drawingOptions_linecolors[indexcolor
                   ]);
            }
            h->GetYaxis()->SetTitle("x␣(cm)");
            h->GetYaxis()->SetTitleOffset(1.35);
            h->SetStats(0);
        }
        g->Draw("p");
        indexcolor++;
    }
    TMarker *vertex = new TMarker(0,0, 21);
    vertex->Draw();
    leg->AddEntry(vertex, "Vertex", "p");
    gPad->ls();
    leg->Draw();
    c1->Modified();
    gSystem->ProcessEvents();

    char saveToFile[100];   // array to hold the result.
    strcpy(saveToFile, saveFolder);
    strcat(saveToFile, Form("/xAxis_event_%d_uid_%d.png", eventId,
       rand()));
    c1->SaveAs(saveToFile);
    return;
}
```

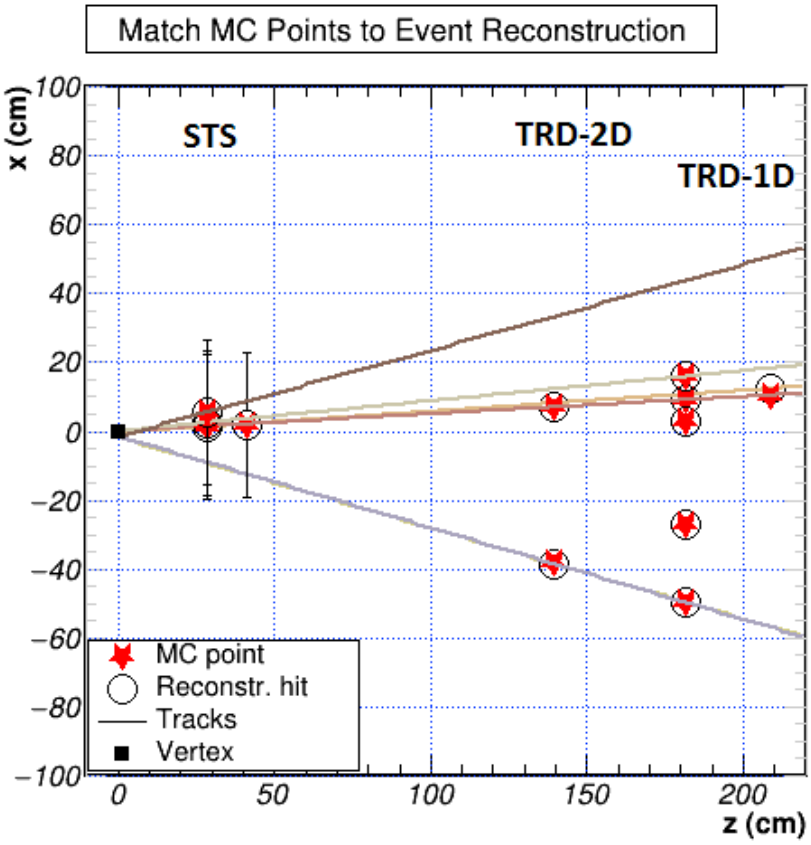


Figure 2.21: Event reconstruction matched to MC points

In addition to providing help in the optimization of event selection and track reconstruction algorithms, this macro shows the positive impact that the TRD-2D has on the vertex reconstruction; when adding the data from the TRD-2D, and drawing those straight line trajectories between hits, the vertex can be reconstructed with greater precision.

Chapter 3

Conclusions

This thesis presents the research and development I carried out within the framework of collaboration for the Compressed Baryonic Matter experiment (CBM) and in particular, for the Transition Radiation Detector 2D (TRD-2D).

The CBM is a hadronic physics experiment which is being constructed at the Facility for Antiproton and Ion Research (FAIR) in Darmstadt, Germany with the aim to study rare observables in the laboratory, by recreating the same density conditions as are seen in Neutron stars. Since those rare probes that scientists are looking for only appear once in a million particle collisions, the CBM will work at very high interaction rates, the highest used thus far in any existing experiment. The CBM will be used to bombard a fixed target with matter at a rate of up to 10MHz and the particles produced in those interactions will pass through a series of detectors which will collect the PB of data. All of the detectors will be self-triggered, meaning that they are online all the time, reporting on any electrical charge they can sense.

The first beam for the CBM experiment is planned for 2026 but until then a prototype - the mini CBM (mCBM) has already been built and is used for data acquisition as well as calibrating and testing the experimental setup. The mCBM accommodates a series of detectors, built by different teams from Europe and Asia and is a collaboration of scientist, engineers and students. One of the detectors in the experiment is the TRD-2D which was designed and built exclusively by the Hadronic Physics Department of the National Insitute for Physics and Nuclear Engineering, Măgurele, Romania. My work for the CBM and mCBM experiments has concentrated on the TRD-2D.

I have worked mostly on running the simulation algorithms (macros) and designing, implementing, testing and validating new macros for interpreting and representing the data visually and also for testing the existing algorithms.

Firstly, I have designed and implemented a macro which mimics the usage of the triggered acquisition model for CBM. The triggered acquisition system is a concept used by most other experiments and infers choosing one of the detectors as a trigger, while all others are latent. Once the reference detector is hit by a particle, it signals to all other detectors to "wake up and listen". I have shown that, while this acquisition model works perfectly in light systems (systems in which one interaction generates only a few particles), it is impossible to use in heavy systems like CBM, where heavy ions such as gold or uranium are used, generating a high multiplicity, at a never before used interaction rate. In a light system, the number of hits

from another detector that are not taken into account because the reference detector missed those particles and thus was unable to send the signal to the other detectors to wake up, is relatively small - so small in fact that the plot showing actual hits and reconstructed hits in such a system, looks very clean and linear. In a heavy system on the other hand, there is an abundance of missed hits. The plots I have generated thus show the exact motivation for which CBM uses a trigger-less free-streaming acquisition model in which all the detectors are online all the time and each one registers and reports the hits independently.

I further investigated the time difference between hits in the TRD-2D and the ToF detectors (first on the information I obtained from simulations, then from real data obtained at mCBM) and also created a plot in which I have compared these time difference histograms (the ones created from simulations vs. real data). I could show that the time difference between hits in the two detectors is negligible - both histograms show a spike in occurrences around the zero mark, but I have also highlighted two issues that appear in simulations - one is a factor 10 difference in S/N ratio and the other is the presence of a spurious component. Both issues require corrections in the simulation code and may well be related to a configuration of the FEE setup and response.

Lastly, it is very important to have a tool for assessing track reconstruction and that is what I have designed and coded using simulation data. I have implemented a macro, to reconstruct the tracks between detector hits (using Monte-Carlo points). The plot obtained using this macro shows the efficiency of vertex reconstruction, once the data from the TRD-2D detector is also taken into account.

The framework I have worked with is a C++ framework originally created for the experiments at CERN, that is now used by an overwhelming number of people in science. It is a framework that facilitates the analysis and processing of big data.

The simulations for mCBM and TRD-2D in particular have to be refined and corrected in order for them to accurately represent the experimental truth, the results we have so far though, are looking promising.

Over the past months I have learnt a lot about the experiment as a whole and the TRD-2D detector in particular, have had successes and failures, but have grown fond of the CBM and it's people and gotten wiser in the process.

List of Figures

Listings

Bibliography

[1] Florian Uhlig for 22nd CBM Collaboration Meeting Dubna. "Introduction to Reconstruction within FairRoot Part 1". In: Sept. 2013. URL: https://redmine.cbm.gsi.de/attachments/19.

[2] Valentina Akishina. "Four-dimensional event reconstruction in the CBM experiment". 2016. URL: https://indico.gsi.de/event/6198/contributions/28511/attachments/20671/26101/Akishina_thesis_corrected.pdf.

[3] Paola Sala Alfredo Ferrari. "Nuclear Reactions in Monte Carlo Codes". In: (Feb. 2002). DOI: 10.1093/oxfordjournals.rpd.a006788.

[4] Volker Friese and. "The high-rate data challenge: computing for the CBM experiment". In: *Journal of Physics: Conference Series* 898 (Oct. 2017), p. 112003. DOI: 10.1088/1742-6596/898/11/112003. URL: https://doi.org/10.1088/1742-6596/898/11/112003.

[5] Etienne Bechtel. "Development of a detector simulation and reconstruction for the CBM-TRD and the analysis of thermal dielectron pairs in 12 A GeV Au+Au collisions". 2020. URL: https://www.uni-frankfurt.de/96680075/Doktorarbeit_Etienne_Bechtel.pdf.

[6] Alexandru Bercuci. *The TRD TDR Addendum - TRD-2D for CBM*. 2021. URL: https://indico.nipne.ro/event/159/contributions/202/attachments/146/227/TRD-TDR-Addendum_TRD2D_ECE-ECSG_211127.pdf.

[7] Pedro Bicudo, Melanie Cardoso, and Nuno Cardoso. "QCD confinement and chiral crossovers, two critical points". In: (Feb. 2011).

[8] Amy Bilton. Dec. 2020. URL: https://home.cern/news/news/knowledge-sharing/rooted-society.

[9] Philipp Kähler for the CBM Collaboration. *The Compressed Baryonic Matter (CBM) Experiment at FAIR*. 2018. URL: https://indico.cern.ch/event/656452/contributions/2869656/attachments/1650146/2638768/CBMatFAIR_PhilippK.pdf.

[10] Volker Friese for the CBM Collaboration. URL: https://indico.cern.ch/event/587955/contributions/2935804/attachments/1679321/2697715/CHEP18_Friese_v2.pdf.

[11] David Emschermann. *The readout system of the CBM experiment*. 2021. URL: https://indico.phy.ornl.gov/event/112/contributions/566/attachments/492/1342/20211208_169_sro9_cbm_daq_v02.pdf.

[12] Volker Friese. *CBM: Experiment, Physics and Trigger*. 2018. URL: https://indico.gsi.de/event/6933/contributions/31365/attachments/22479/28222/TDHEP2-Friese.pdf.

[13] *GEANT4 simulation toolkit*. URL: https://geant4.web.cern.ch/.

[14] *GSI - FAIR*. URL: https://www.gsi.de/en/researchaccelerators/fair.

[15] Simon Christian Helmut. "Investigations of rate and multi-hit capability of multi-gap resistive plate chambers". Feb. 2021. DOI: 10.11588/heidok.00029376.

[16] Robert Kwiatkowski. *Monte Carlo Simulation — a practical guide*. URL: https://towardsdatascience.com/monte-carlo-simulation-a-practical-guide-85da45597f0e.

[17] Axel Puntke. "First mTRD Performance Studies in the mCBM 2020 Campaign". 2021. URL: https://indico.gsi.de/event/13609/contributions/58014/.

[18] *ROOT manual - Histograms*. URL: https://root.cern/manual/histograms/.

[19] Shreya Roy. *Time based simulations for Ni+Ni setup*. May 2022. URL: https://indico.gsi.de/event/15139/contributions/63612/attachments/39860/54392/Time%20based%20simulations%20for%20Ni%2BNi%20setup.pdf.

[20] Peter Senger. "Astrophysics in the Laboratory—The CBM Experiment at FAIR". In: *Particles* 3.2 (2020), pp. 320–335. ISSN: 2571-712X. DOI: 10.3390/particles3020024. URL: https://www.mdpi.com/2571-712X/3/2/24.

[21] Adrian Weber. *The mCBM experiment at SIS18 of GSI/FAIR - a CBM precursor and demonstrator*. 2022. URL: https://indico.cern.ch/event/895086/contributions/4724533/.