1 2 9 0

UNIVERSIDADE Đ COIMBRA

João Fernando Marques da Silva

DEVELOPMENT OF THE UNDERGROUND PERFORMANCE MONITOR FOR LZ

Dissertação no âmbito do Mestrado Integrado em Engenharia Física orientada pelo Professor Doutor Francisco Neves e pelo Professor Doutor Vladimir Solovov e apresentada ao Departamento de Física da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Outubro de 2020

1 2 9 0

UNIVERSIDADE Ð COIMBRA

João Fernando Marques da Silva

Development of the Underground Performance Monitor for LZ

Thesis submitted to the
University of Coimbra for the degree of
Master in Engineering Physics

Supervisors:
Professor Francisco Neves (LIP - Coimbra)
Professor Vladimir Solovov (LIP - Coimbra)

Coimbra, 2020

This work was developed in collaboration with:

Faculdade de Ciências e Tecnologia da Universidade de Coimbra

LIP – Coimbra

LUX-ZEPLIN

Acknowledgments

I would like to start by thanking the elements of jury: Professor Doctor Vitali Tchepel, Doctor Andrey Morozov, Doctor Francisco Neves and Doctor Márcio Ferreira, for the effort of reading this rather long dissertation. I hope you will find it interesting and insightful.

I will now briefly switch to Portuguese for more personal acknowledgments.

Gostaria de agradecer a todos os elementos do *Dark Matter Group* do LIP-Coimbra por me terem acolhido durante o último ano e por me terem feito sentir em casa. Em especial, quero agradecer aos meus orientadores Francisco Neves e Vladimir Solovov por terem estado sempre disponíveis para me ajudar, e pelas muitas horas que passaram a ler e a reler este trabalho.

Aos amigos que fiz durante os últimos 5 anos da minha vida académica. À Fátima, companheira no gabinete e no gosto por pastéis de nata. Ao Manuel, mestre dos lanches matinais e (sub)capitão dos Presuntos FC. À Maria, desenhadora de cadernos profissional e fonte infindável de *memes*. E, finalmente, ao meu colega João Frazão, um primaço desde o primeiro dia que nos conhecemos. A todos vocês um muito obrigado por terem estado sempre do meu lado nos bons e nos maus momentos. Ao Frazão e à Maria ainda um obrigado muito especial por me terem ajudado a rever este trabalho.

Aos meus amigos mais antigos: Henrique, João Paulo e Luis, quero agradecer pela quantidade de anos que me aturam e por serem a fonte de muitos momentos que vão ficar para a história.

Por fim, quero agradecer aos meus pais por me terem tornado na pessoa que sou hoje. Por todo o apoio que sempre me deram e por nunca me terem pressionado em nenhuma altura do meu percurso académico. Sem vocês nunca teria conseguido chegar aqui.

Resumo

A experiência LUX-ZEPLIN (LZ) faz parte da nova geração de experiências de deteção direta de WIMPs, que continuam a ser um dos candidatos mais bem posicionados para explicar o mistério da Matéria Escura. O detetor LZ é uma TPC com 10 toneladas de Xénon líquido, o que faz desta experiência a maior alguma vez construída utilizando esta tecnologia. Juntamente com a TPC, dois detetores adicionais (*Skin e Outer detectors*) são utilizados, de forma a reduzir e a rejeitar os eventos de fundo. A combinação dos três detetores irá permitir a LZ atingir uma sensibilidade de 2×10^{-48} cm^2 para interações WIMP-nucleão independentes de spin.

A complexidade da experiência cria a necessidade de monitorizar em tempo-real os detetores, de forma a garantir o seu normal funcionamento bem como a qualidade dos dados adquiridos. O Underground Performance Monitor (UPM) é o subsistema de LZ, instalado localmente, desenvolvido especificamente para esta tarefa. A sua implementação atual, funcionando apenas em CPUs, não atinge a taxa de processamento necessária para monitorizar os detetores de LZ em tempo-real, durante as aquisições para calibração.

Este projeto propõe utilizar aceleração por GPUs como solução para este problema. O objetivo é implementar em GPU alguns dos algoritmos da cadeia de análise do UPM. Foram observadas pequenas vantagens no desempenho de três dos algoritmos implementados: normalização de *waveforms*, soma de *waveforms* e parametrização de pulsos. Um grande aumento de desempenho ($\sim$30$\times$) foi alcançado para o algoritmo de reconstrução da posição.

Esta dissertação contém uma introdução ao problema da Matéria Escura e ao funcionamento de LZ, bem como uma visão geral de programação em GPU utilizando CUDA. Uma descrição completa dos algoritmos desenvolvidos também está incluída. No final, os resultados são apresentados e discutidos.

Palavras-Chave: LUX-ZEPLIN, GPU, monitorização em tempo-real, UPM, Matéria Escura

Abstract

LUX-ZEPLIN (LZ) is part of the new generation of direct detection experiments aiming at the detection of WIMPs, still one of the best motivated candidates to explain the Dark Matter mystery. The LZ detector is a TPC featuring 10 tonnes of liquid Xenon, making it the largest detector ever built using this technology. Alongside the TPC, a Skin and an Outer detector are also used, to further reduce and reject the background events. The combination of these three detectors will allow LZ to reach an unprecedented sensitivity of 2×10^{-48} cm^2 for spin-independent WIMP – nucleon interaction.

The complexity of the experiment creates a need for real-time monitoring of the detectors, to ensure their performance and the quality of the data being acquired. The Underground Performance Monitor (UPM) is the onsite LZ subsystem designed specifically for this task. Its current implementation, running solely on CPUs, does not reach the necessary processing rate, during calibration runs, to monitor in real-time the LZ detectors.

This project proposes to use GPU-acceleration as a solution for this problem. The objective is to implement on GPUs some of the algorithms of the UPM analysis chain. Small performance advantages were observed for three of the implemented algorithms: waveform normalization, waveform sum and pulse parameterization. A major performance increase ($\sim 30\times$) was achieved for the position reconstruction algorithm.

This dissertation contains an introduction to the Dark Matter problem and to the functioning of LZ as well as an overview of GPU programming using CUDA. A full description of the developed algorithms is also included. Finally, the results are presented and discussed.

Keywords: LUX-ZEPLIN, GPU, real-time monitoring, UPM, Dark Matter

Acronyms and Abbreviations

ADC – Analog to Digital Converter
CPU – Central Processing Unit
CUDA – Compute Unified Device Architecture
DAQ – Data Acquisition
DtoH – Device to Host
EB – Event Builder
GPU – Graphics Processing Units
HG – High Gain
HtoD – Host to Device
LG – Low Gain
LXe – Liquid Xenon
LZ – LUX - ZEPLIN
LZap – LZ analysis package
OD – Outer Detector
PMT – Photomultiplier Tube
POD – Pulse Only Digitization
RC – Run Control
SC – Slow Control
SM – Streaming Multiprocessor
TBA – Top-Bottom Asymmetry
TPC – Time Projection Chamber
UPM – Underground Processing Monitor
WIMP – Weakly Interactive Massive Particle

List of Figures

List of Tables

Contents

1

Introduction

1.1 LUX - ZEPLIN

The LUX – ZEPLIN (LZ) Dark Matter experiment is an international collaboration that resulted from merging the former LUX and ZEPLIN collaborations. It comprises 37 institutions from 5 different countries, of which Portugal and LIP – Coimbra are part.

Installed approximately 1500 m underground at the Sanford Underground Research Facility (SURF) in Lead, South Dakota, USA, LZ is part of the new generation of experiments aiming at the direct detection of WIMPs (Weakly Interactive Massive Particles), one of the best motivated candidates to explain the mystery of Dark Matter (DM). The LZ detector is a dual-phase time projection chamber (TPC) featuring 10 tonnes of liquid Xenon (LXe) and it will be the largest detector ever built using this technology. Inside the TPC there are 494 PMTs divided in two arrays (241 PMTs for the bottom array and 253 for the top). Alongside the TPC, two additional detectors are used in LZ: a Skin and an Outer Detector (OD). The Skin is a layer of LXe made of 131 PMTs and located between the PTFE-clad field cage and the inner cryostat wall. The OD is a gadolinium-loaded liquid scintillator (GdLS) that surrounds the cryostat and is made of 120 PMTs (see Section 2.6 and Fig. 2.11 for more details about the detectors) [1].

The combination of the mass of LXe used, with the extra capability of tagging external and internal backgrounds given by the Skin and the OD, results in an expected sensitivity of 2×10^{-48} cm^2 for spin-independent WIMP-nucleon interactions, during its 3-year run, making LZ the most sensitive experiment to date [1].

To operate, monitor and analyze the data acquired by the experiment a number of subsystems are being developed, both onsite and offsite.

The systems used for monitoring and operating LZ are installed onsite. These systems are: Run Control, Slow Control, Data Acquisition, Event Builder and Underground Performance Monitor. Installed offsite are the data centers, where the offline analysis takes place. The Run Control (RC) is the subsystem responsible for coordinating the operation of the entire experiment. It determines the type of run being conducted (calibration, WIMP search, etc.) and communicates with all the other onsite subsystems to configure them accordingly. The Slow Control (SC) is designed to ensure that the experiment is operating safely by monitoring and supervising all major LZ subsystems. Data Acquisition (DAQ) is responsible for digitizing the amplified and shaped signals from the PMTs. The Event Builder (EB) takes the data from the DAQ and saves it to disk, in a format used by both the online and offline analysis [2].

The offline analysis is done in the aforementioned data centers. LZ will use two data centers, one in the UK and one in the USA, which will run the LZ analysis package (LZap) [2]. These data centers are large clusters of computers and possess an enormous amount of processing power. Despite this fact, they are unable to provide a real-time monitoring of the LZ experiment due to the time it takes to transfer data over the internet to and from SURF, in addition to the queuing of jobs at the data centers ($\sim$4/8 hours in total). As a result, a need for a real-time monitoring subsystem installed onsite emerged and the solution found was the Underground Performance Monitor (UPM).

1.2 UPM

The UPM is the LZ subsystem installed onsite responsible for the real-time monitoring of the TPC, OD and Skin detectors. It will perform an analysis of the raw[1] detector data, oriented towards detector performance and fault detection.

It communicates with other LZ subsystems like the RC, the SC and the EB. The RC indicates to the UPM when the EB has files that are ready for analysis. It also tells the UPM which configuration file is to be loaded from the LZ configurations database. This file contains the parameters that are used by the UPM algorithms. After running the analysis algorithms and processing the data, the UPM sends to the SC sets of parameters indicating the status of the detector, such that the SC can trigger any necessary alarms. The processed data is also made available in a HTTP viewer, to allow human operators to easily visualize the results of the analysis and verify how the detector is performing [2].

The largest constraint for the real-time monitoring capabilities of the UPM is

[1]unprocessed

related to the amount of CPUs that can be installed in the limited space available underground. Despite this limitation, the UPM, as it is developed at the moment, satisfies the requirements for real-time monitoring for WIMP search runs. However, the same is not true for calibration runs, which require a higher processing rate to deal with the incoming events.

So, as a part of its participation in the LZ experiment, LIP – Coimbra proposed to test the usage GPU-acceleration to improve the performance of the UPM and thus meet the requirements for real-time monitoring for all types of acquisition runs.

1.3 Objective

The objective of this project is to develop a GPU-based analysis framework, which will be integrated into the already existing UPM framework. The targeted algorithms are the ones that are part of the signal processing chain already implemented for CPUs: waveform normalization and sum, pulse finding, pulse parameterization and position reconstruction.

1.4 GPUs

Graphics processing units (GPUs) are specialized electronic circuits designed specifically for image processing and graphical display in most devices (computers, mobile phones, gaming consoles, etc.). The first device that could be properly called a GPU was the NEC μPD7220 developed in the 1980s. Since then a lot of development has been directed towards these devices, namely by Intel, NVIDIA and AMD, the current major market share leaders. Intel leads the market of integrated graphics solutions, while NVIDIA and AMD lead the market of dedicated graphics cards.

In the last 20 years, GPUs have evolved from being solely graphical display devices into general programming devices. This new paradigm of GPU utilization is called General Purpose GPUs (GPGPU). This evolution allowed GPUs to be used to perform computationally intensive tasks that were previously done on the CPU, and thus improve the performance of an application.

Applications that use CPUs and GPUs to perform its computations are commonly referred to as GPU-accelerated. The general idea of GPU-acceleration is to offload the more computationally intensive parts of a program to the GPU, and keeping the other sections running on the CPU. This takes advantage of the excellent parallel computing characteristics of the GPUs, and uses them to speed up the computation of a task.

A common, and simple, example of GPU-acceleration is the sum of two arrays. Whereas with CPUs, elements needed to be summed one by one in a sequential manner, with GPUs, batches of elements can be summed all at once, with the number of elements per batch being related to the maximum number of threads that can be executed concurrently by the GPU.

As of now, the most common languages for GPU programming are OpenCL and CUDA. For this project NVIDIA GPUs will be used and the programming language chosen is CUDA, since it is developed specifically for NVIDIA devices.

1.5 Structure

This dissertation is structured in the following form. Chapter 2 will introduce the concept of Dark Matter by presenting an overview of the evidences present in the Universe, as well as giving insight as to what WIMPs are expected to be. A brief mathematical explanation of the ΛCDM model is also given, before discussing the current state of Dark Matter searches, with a special emphasis on the characteristics of LUX-ZEPLIN.

In Chapter 3, GPUs and the CUDA programming model will be characterized. Then an overview of the current state of GPU-based applications in different areas of physics is presented.

Chapter 4 delves further into the UPM, and discusses the details of all the algorithms developed for the GPU-framework.

The tests performed on the developed algorithms are going to be presented on Chapter 5. This chapter will also contain the obtained results and their discussion.

Finally, Chapter 6 will serve as a round-up of this dissertation, summarizing the conclusions taken from this project as well as discussing possibilities of further development.

2

Dark Matter

Since the beginning of the 20th century, observations of large cosmological structures indicated that the visible mass of those structures only accounted for a small fraction of their total mass, and that the bigger fraction of the mass in the system was not detectable. So, to solve this “missing mass” problem, the concept of Dark Matter was postulated as a mysterious and unseen type of matter that does not absorb, emit or scatter light [3].

The first person to propose the existence of Dark Matter was Fritz Zwicky[1], in order to explain the discrepancies between the mass calculated by the galactic motion and the one deduced by the visible matter for the Coma cluster. He applied the Virial theorem to estimate the mass of this cluster, obtaining a value of $M_c = 4.5 \times 10^{13} M_{\odot}$, which is approximately 170 times greater than the value that can be inferred by luminosity alone. More recent approaches to this problem account for the interstellar gas and use more precise values of the Hubble constant, giving a mass of $M_c = 2 \times 10^{15} M_{\odot}$, with the vast majority of it being Dark Matter [4].

2.1 Evidences of Dark Matter

The following sections will present various evidences of the existence of Dark Matter in the Universe. By studying these evidences it is possible to establish various constraints on the nature of Dark Matter and, more generally, on the mass-energy content of the Universe. These constraints fit into a larger cosmological model, the Lambda Cold Dark Matter (ΛCDM) model, which will be presented in Section 2.2).

[1]a Swiss astronomer

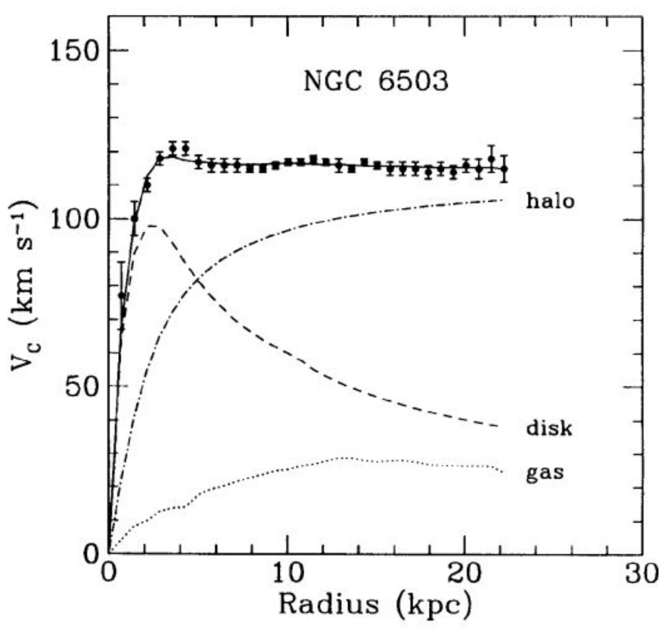


Figure 2.1: Rotation curve of the galaxy NGC 6503. The full line represents the best fit for the velocity data obtained. The dashed and dotted lines represent the contributions from the mass density of the gas and the galactic disk, and the predicted contribution from the mass density of the Dark Matter halo [6].

2.1.1 Dynamical behaviour of galaxies and clusters

The first evidences of Dark Matter came from the dynamical behaviour of galaxies and galaxy clusters. It was expected that the bodies further away from the center of a galaxy should move slower than the ones closer to it, as per the law of gravitation set by Newton (eq. 2.1)

$$v(r) = \sqrt{\frac{GM(r)}{r}} \tag{2.1}$$

where r is the radial distance, v(r) is the velocity, M(r) is the mass contained in a shell of radius r and G the gravitational constant.

However, by analyzing rotational curves of various galaxies (Fig. 2.1) it can be perceived that, rather than falling down, the velocity of stars and gas clouds at large radii remained roughly constant[2]. This phenomenon can be explained by the presence of a Dark Matter halo that reaches beyond the luminous disk of the galaxy [5].

[2]this was first observed by Vera Rubin, an American astronomer

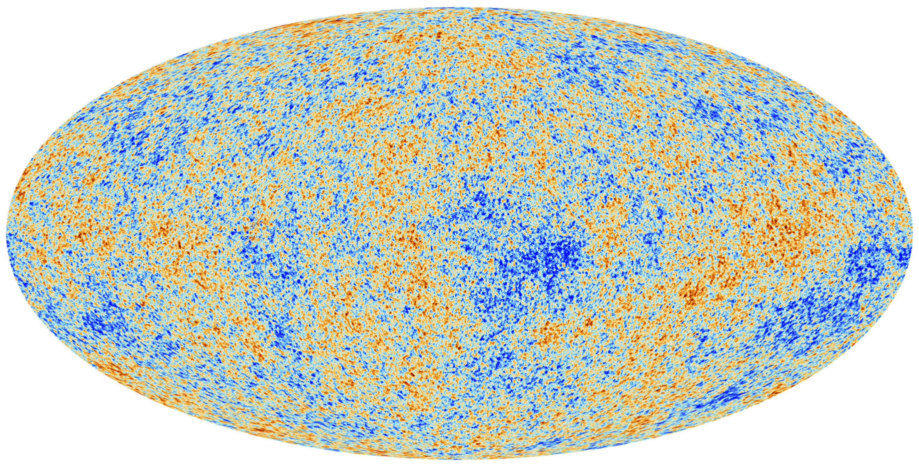


Figure 2.2: The anisotropies of the Cosmic Microwave Background as observed by the Planck collaboration.©ESA

2.1.2 Cosmic Microwave Background

The Cosmic Microwave Background (CMB) is electromagnetic radiation that is remnant of the primordial stages of the Universe. In these initial stages a photon-baryon-electron plasma was created. Inside this plasma the temperatures were very high and, due to photo dissociation, the creation of neutral hydrogen was not possible [7].

As the Universe continued to expand, the temperature decreased to a point where protons and electrons combined, forming electrically neutral hydrogen. At this point, the energy of the photons was no longer enough to ionize the hydrogen, and so their interaction stops, leaving the photons to roam across the Universe. The wavelength of these photons increased with the expansion of the Universe and is now on the microwave region, constituting the CMB [7].

Since the CMB is sensitive to most of the ΛCDM model parameters (see Section 2.2), its analysis allows to calculate those parameters and determine the constitution of the Universe in terms of its components (baryonic matter, dark matter, energy, etc.).

2.1.3 Baryon Acoustic Oscillations

In the embryonic stages of the Universe, denser regions gravitationally attracted more matter. This gravitational attraction compressed matter, causing the production of photons due to the increase of temperature. These photons exerted an outward pressure that only affected baryonic matter, due to the tight coupling be-

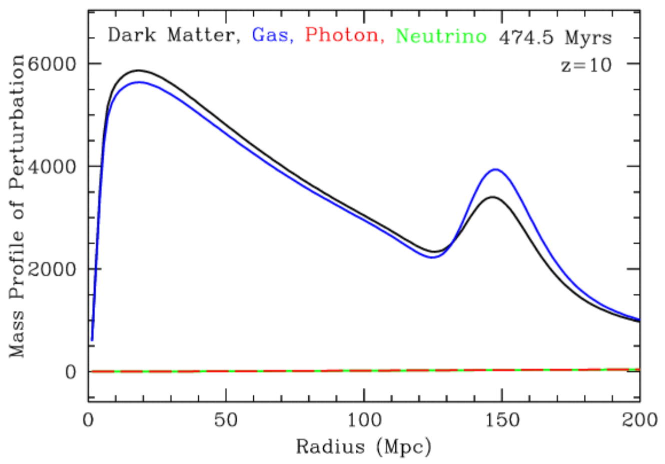


Figure 2.3: Representation of the mass profile of an acoustic perturbation. The peak at 150 Mpc is the sound horizon [10].

tween the two. The counteraction between this pressure and the gravitational forces created oscillations, called Baryon Acoustic Oscillations (BAO) [5].

The baryon-photon acoustic wave created by the outward pressure moved away from the density center, leaving the Dark Matter behind since it did not interact with the photons.

When the temperature became low enough for the photons and baryons to decouple, the BAO stalled and a shell of baryonic material was deposited at a defined radius (called sound horizon), while the photons continued their movement unobstructed. Because of this, it is expected that the greater part of the massive structures observed today are separated by a distance equal to the sound horizon [8]. By measuring the abundance of intergalactic gas by redshifted Lyman-α emissions, the sound horizon length scale can be detected [9].

Like CMB, BAO also allows to calculate some cosmological parameters (namely the density of baryonic matter Ω_b), since it is sensitive to the acceleration rate of the Universe and to the baryonic fraction of its total mass [8].

2.1.4 Big Bang Nucleosynthesis

Big Bang Nucleosynthesis (BBN) was the production of light elements (H, D, ^{3}He, ^{4}He, ^{6}Li and ^{7}Li) that happened in a short time window right after the Big Bang, when the temperature of the proton-neutron plasma was of the order of 10 MeV [11]. The relative abundance of light elements we see today is a result of the BBN.

By comparing the relative abundance of the mentioned elements with the one expected by studying the BBN it is possible to estimate the value of the baryon-to-photon ratio η, and calculate the baryonic energy density parameter [11].

Having this energy density allows to estimate the amount of visible matter present in the Universe, and from that also estimate the amount of dark matter.

2.1.5 Type Ia Supernova

A Supernova is a massive explosion of a star. There are two major types of supernovas. The first one happens at the end of the lifetime of a massive star (M $>\sim$8 M$_\odot$). As the star starts running out of nuclear fuel its own mass flows into the core, up to a point where it cannot withstand its own gravitational force causing it to collapse, resulting in a large explosion [12].

The second type of supernova happens in binary systems, in which one of the stars is a white dwarf. A white dwarf is a small and very dense star that is stable as long as its mass stays below a limit of 1.4 M$_\odot$, known as Chandraseckhar limit [12]. If the white dwarf is part of a binary system it can accumulate matter from its companion star and its mass would grow, eventually reaching the aforementioned limit. Once above the Chandraseckhar limit, the stability of the white dwarf can no longer be maintained by the electron degeneracy pressure[3], resulting in a violent thermonuclear explosion of its carbon-oxygen core. This type of supernova is known as Type Ia Supernova. Since this type of supernova always happens for a precise mass of the white dwarf, its luminous output is roughly the same between events. Therefore, these events can be used to determine cosmological parameters, and to look at the acceleration history of the Universe [13].

Eq. 2.2 relates the distance modulus μ to the redshift z and the deceleration parameter q_0. The quantity d_L is the luminosity distance of the observed object. Studies of Ia Supernovas give a value of $q_0 \approx -0.55$ for the deceleration parameter, which means that the expansion of the Universe is speeding up. This value also indicates that the total energy density of the Universe is dominated by dark energy.

$$\begin{aligned}\mu =& 5\log_{10}\left(\frac{d_L}{10pc}\right) \approx \\ \approx& 5\log_{10}(z) + 1.086(1-q_0)z - 5\log_{10}(h) + 42.40 + O(z^2)\end{aligned} \tag{2.2}$$

[3]As the original star is collapsing into a white dwarf its lowest electron energy levels are filled. Due to the Pauli exclusion principle, some of those electrons are then forced to occupy higher energy levels, creating an outwards pressure that balances the effect of the gravitational force and prevents further collapsing.

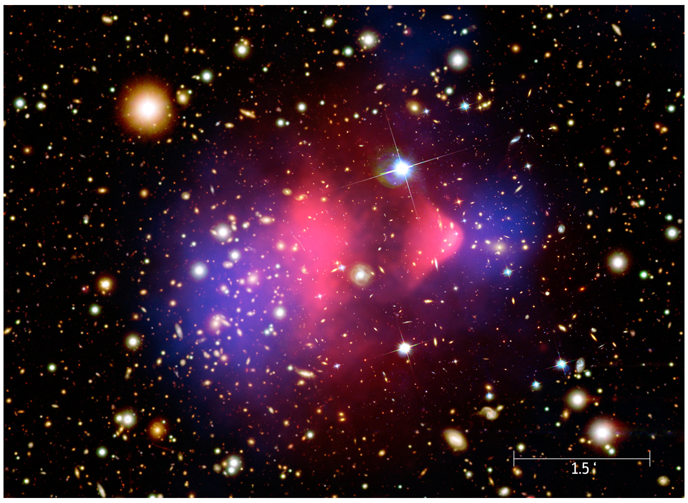


Figure 2.4: Image from the Bullet Cluster where the contributions from the intergalactic gas (pink) and the majority of the matter (blue) are represented. The contribution of the intergalactic gas was inferred from observing the X-ray emissions of the cluster, while the contribution from the majority of the matter was inferred from weak gravitational lensing. (Figure credit: X-ray: NASA/CXC/CfA/ M.Markevitch et al.; Lensing Map: NASA/STScI; ESO WFI; Magellan/U.Arizona/ D.Clowe et al. Optical image: NASA/STScI; Magellan/U.Arizona/D.Clowe et al.).

2.1.6 Gravitational Lensing

The presence of a massive object can distort the space-time, thus bending the trajectory of light in an effect called gravitational lensing. There are two main regimes that can occur, strong and weak gravitational lensing.

The first regime happens when the light is distorted in a way that significant aberrations (e.g. arching) are observed. In the second regime the light distortion is very slight and it can only be perceived trough statistical analysis.

There is also a third, very rare, regime called microlensing. It occurs when a massive object is aligned with a bright light source increasing the luminosity of the source without distorting it. The rareness of this effect comes from the low probability of such an alignment to occur [5].

Gravitational Lensing is an important tool for the detection of Dark Matter, because it allows the detection and measurement of the mass of massive distributions with little (or no) light emission, as is the case for DM [14].

Collisions between galaxy clusters are perhaps the most common case of using weak gravitational lensing to find evidences of Dark Matter. The Bullet Cluster is the most well known example of this (Fig. 2.4). The observation of its collision showed a clear offset between the position of the baryonic matter of the cluster, and the majority of the matter which generated the observed gravitational lensing.

Moreover, this offset showed that the majority of the matter present in the colliding clusters did not significantly interact with baryonic matter nor with itself, a characteristic that is fundamental in the definition of Dark Matter [14].

2.2 The ΛCDM model

The Lambda Cold Dark Matter (ΛCDM) model is a mathematical parameterization of Big Bang cosmology. This model assumes a cosmological constant Λ and a non-baryonic Cold Dark Matter component. The parameters of this model allow the description of a number of cosmological properties such as the expansion of the Universe, the observed abundances of matter, the underlying structure behind galaxy distribution and the existence of the CMB [11].

The expansion of the Universe was first observed by Edwin Hubble, by measuring the redshift (z) of several galaxies.

$$cz = H_0 r \tag{2.3}$$

Eq. 2.3 represents the relation between the measured redshift and the distance r to the galaxy. H_0 is the Hubble constant and represents the rate of expansion at the current time t_0, and c is the speed of light in vacuum. By analyzing this equation it is possible to conclude that, as the distance to the observed galaxy increases, so does the measured redshift.

In fact, the Hubble parameter is only constant at a specific time. To calculate its value at a given time t, eq. 2.4 needs to be used.

$$H(t) = \frac{\dot{a}(t)}{a(t)} \tag{2.4}$$

The cosmic scale factor $a(t)$ is a representation of the size of the Universe at time t. This factor can be related with the energy density $\epsilon(t)$, the curvature parameter k and the pressure $p(t)$ by the Friedmann equations (eqs. 2.5 and 2.6), where R_0 is the curvature radius if $k \neq 0$ [7].

$$\left(\frac{\dot{a}}{a}\right)^2 = H^2 = \frac{8\pi G}{3c^2}\epsilon - \frac{kc^2}{R_0 a^2} + \frac{\Lambda}{3} \tag{2.5}$$

$$\frac{\ddot{a}}{a} = -\frac{4\pi G}{3c^2}(\epsilon + 3p) + \frac{\Lambda}{3} \tag{2.6}$$

These equations are valid to describe the expansion of the Universe, assuming that the Cosmological Principle[4] is true.

The cosmological constant $\Lambda = 4\pi G\epsilon/c^2$ was an *ad hoc* solution proposed by Einstein in order to counteract the attractive effect of gravity, and to obtain a static Universe ($\ddot{a} = 0$). This proposed solution was later found to be the vacuum energy density (also known as Dark Energy) [11].

The contribution of Λ can be included in the energy density parameter, so that ϵ accounts for all the different energy contributions of the Universe. The density parameter is then defined by eq. 2.7

$$\Omega(t) = \frac{\epsilon(t)}{\epsilon_c(t)} \tag{2.7}$$

where $\epsilon_c(t)$ is the critical energy density of the Universe, calculated for a flat Universe, $k = 0$ (eq. 2.8) [7]

$$\epsilon_c(t) = \frac{3c^2}{8\pi G} H(t)^2 \tag{2.8}$$

The density parameter can be used to represent the first Friedmann equation (eq. 2.5) in a different form:

$$\frac{H^2}{H_0^2} = \frac{\Omega_r}{a^4} + \frac{\Omega_m}{a^3} + \frac{\Omega_k}{a^2} + \Omega_\Lambda \tag{2.9}$$

Here the parameters Ω_r, Ω_m, Ω_k and Ω_Λ represent the energy density for radiation, matter, curvature of space and vacuum energy at time t_0, respectively. At time t_0 the scale factor a is set to 1, so the sum of the different density contributions will add up to 1 (eq. 2.10) [7]

$$\Omega_r + \Omega_m + \Omega_k + \Omega_\Lambda = 1 \tag{2.10}$$

By analysing eq. 2.5 for time t_0, it is possible to find a relation between the parameter Ω_k and the current energy density parameter Ω_0:

$$\Omega_k = 1 - \Omega_0 \tag{2.11}$$

This means that by measuring Ω it is possible to get a measurement of the curvature of space-time: $\Omega = 1$ results in a flat Universe, $\Omega > 1$ results in a closed Universe and $\Omega < 1$ results in an open Universe.

The contributions from matter and vacuum energy can be used to measure the

[4]The Cosmological Principle states that the distribution of matter in the Universe is isotropic and homogeneous at a certain scale

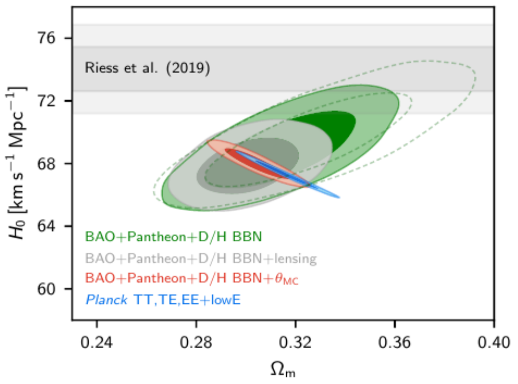


Figure 2.5: Inverse-distance-ladder constraints on the Hubble parameter and Ω_m in the base ΛCDM model. Here the results from CMB (Planck TT, TE, EE+lowE), BAO, type Ia supernovas (Pantheon), BBN and gravitational lensing are used together in different combinations originating different constraints [15].

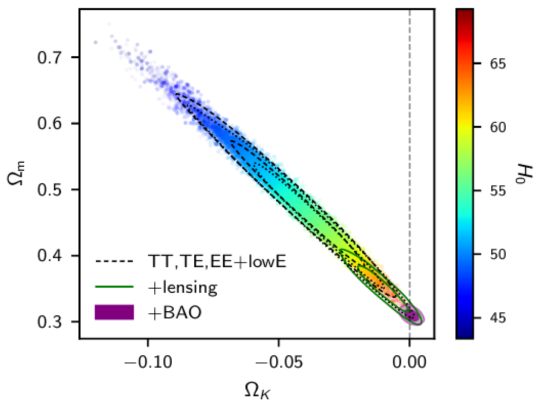


Figure 2.6: Constraints on a non-flat Universe as a minimal extension to the base ΛCDM model. The constraints obtained by joining CMB data (TT, TE, EE+lowE), gravitational lensing and BAO data are shown. The full joint constraint of all these evidences is consistent with a flat Universe ($\Omega_k \sim 0$) [15].

deceleration parameter q_0 (eqs. 2.2 and 2.12) [7]

$$q_0 = -\left(\frac{\ddot{a}}{aH^2}\right)_{t=t_0} = \Omega_r + \frac{\Omega_m}{2} - \Omega_\Lambda \tag{2.12}$$

If $q_0 < 0$ the expansion rate of the Universe is speeding up, if $q_0 > 0$ the expansion rate of the Universe is slowing down.

The current best estimates for the parameters of the ΛCDM model are presented in Table 2.1 [15]. These estimates are obtained by joining the constraints derived from observing the various evidences presented in Section 2.1 (Figs. 2.5 and 2.6). Although the individual constraints stem from very different evidence sources, by joining them in different combinations it is possible to see that the joint constraints all overlap in the same region, which allows to get accurate estimates for the parameters of the ΛCDM model [15].

Assuming the ΛCDM model and the best fit values presented on the table, it can be inferred that presently the Universe is flat and matter-dominated (Ω_r and $\Omega_k \sim 0$) (Eqs. 2.9 and 2.10). It is also possible to make an estimation of the mass-energy content of the Universe: 68.47% Dark Energy, 31.53% Matter of which 26.45% is CDM and 4.93% is Baryonic matter [16].

Parameter		**Value**
Age of the Universe [Gyr]	t_0	13.797 ± 0.023
Hubble constant [km s^{-1} Mpc^{-1}]	H_0	67.36 ± 0.54
Baryon energy density	$\Omega_b h^2$	0.02237 ± 0.00015
CDM energy density	$\Omega_c h^2$	0.1200 ± 0.0012
Matter energy density	Ω_m	0.3153 ± 0.0073
Dark energy density	Ω_Λ	0.6847 ± 0.0073

Table 2.1: Estimates for the cosmological parameters of the ΛCDM model obtained by the Planck Collaboration. The values presented have a 68% confidence level [15]. The parameter h is the dimensionless Hubble parameter ($h = H_0/(100\,km\,s^{-1}\,Mpc^{-1})$)

.

2.3 Dark Matter Candidates

There are a number of suggested candidates to explain the mystery of Dark Matter like: Massive Astrophysical Compact Halo Objects (MACHOs), neutrinos, axions or Weakly Interactive Massive Particles (WIMPs), for instance.

Axions and WIMPs are, as of now, some of the candidates that best fit the characteristics assumed by the ΛCDM model for the nature of Dark Matter [17].

2.3.1 Axions

Axions are hypothetical particles postulated by Peccei and Quinn to explain the Charge-Parity (CP) problem in strong interactions. According to Quantum Chromodynamics (QCD) a violation of CP symmetry could occur for strong interactions. However, this breaking of symmetry was never observed experimentally.

This strong CP problem manifests itself by the difference between the expected large value for the neutron electric dipole moment, d_n, and the value that was in fact measured ($|d_n| < 2.9^{-26} e\,cm$). To solve the problem, Peccei and Quinn introduced a global symmetry that could spontaneously break generating a pseudo-Goldstone boson, that would be the axion [18].

The existence of the axion would also contribute to the dark matter density of the Universe because, as they would not be produced thermally, their speed would not reach relativistic levels, making them a part of cold dark matter [19].

The fact that axions couple with magnetic fields allows them to be detected by measuring radio frequencies inside a microwave resonant cavity, with a strong magnetic field applied. The measured frequencies would correspond to the Compton wavelength of the axion and would allow to determine its mass. The best results from the ADMX experiment excluded axions with masses in the 1.9-3.53 μeV range [20].

Experiments aiming at the direct detection of WIMPs are also able to detect axions. This possibility occurs due to the axio-electric effect, a phenomenon where an axion is converted into a photon, generating electron recoils [19]. Earlier this year, the XENON collaboration reported the observation of a 3.5σ excess of the low-energy electronic recoils over the predictions of the background model, which can be attributed to solar axions [21]. However, several alternative explanations are also possible such as unaccounted backgrounds from tritium or ^{37}Ar [22].

2.3.2 WIMPs

WIMPs are considered to be one of the strongest candidates for the explanation of Dark Matter [3]. These are theorized to be neutral particles of non-baryonic nature with a mass in the GeV to TeV range. They do not interact strongly with matter nor through the electromagnetic force. Another important characteristic of WIMPs is their stability in the time scale of the Universe. This factor can explain the current abundance of Dark Matter [3].

The production of WIMPs would have been abundant immediately after the Big Bang. In those extreme conditions they would be in equilibrium with radiation while the temperature exceeded their mass. WIMPs could also be created by interactions between particle-antiparticle pairs that coupled to the weak force [5].

$$\chi + \bar{\chi} \rightleftharpoons X + \bar{X} \tag{2.13}$$

In eq. 2.13 $\chi/\bar{\chi}$ represent the WIMP particle/antiparticle and $X/\bar{X}$ represent any particle/antiparticle pair.

When temperatures dropped bellow the WIMP mass, the production of WIMPs was suppressed and their abundance started to decrease at a rate Γ_A (eq. 2.14)

$$\Gamma_A = n_\chi \langle \sigma_A \nu \rangle \tag{2.14}$$

where n_χ is the WIMP number density, σ_A the WIMP annihilation cross section and ν the relative velocity of the two particles. As annihilation proceeds, n_χ decreases and thus the annihilation rate. When Γ_A decreases to a value below the expansion rate of the Universe, the thermal equilibrium is broken and the reaction in eq. 2.13 stops. The result of this is a thermal remnant density of WIMPs that, if they really prove to be stable over the time frame of the Universe, still remains to this day [3].

An analytical approximation is obtained in [3] for the energy density contribution of WIMPs (eq. 2.15) with an expected error of $\sim 10\%$.

$$\Omega_\chi h^2 \approx \frac{3 \times 10^{-27} cm^3 s^{-1}}{\langle \sigma_A \nu \rangle} \tag{2.15}$$

The energy density contribution obtained for a massive particle with a σ_A of the scale of the weak interaction ($\sim 10^{-25} cm^2 s^{-1}$) is very much similar to the one observed for non-baryonic matter ($\Omega_c h^2$) (see Table 2.1).

As was seen in Section 2.1.1, rotational curves are used to characterize density of matter (baryonic or not) present in a galaxy. The rotational curve of the Milky Way, estimates the Dark Matter density to be between $0.235 \pm 0.030\, GeV/cm^3$ and $0.389 \pm 0.025\, GeV/cm^3$. A standard value of $\rho_0 = 0.3\, GeV/cm^3$ is adopted by most experiments for an easier comparison of the results [23].

The WIMP canonical model assumes a WIMP velocity modeled by a Maxwell-Boltzmann distribution with solar circular velocity of $\nu_0 = 220\, km/s$, cut off at an escape velocity of $\nu_{esc} = 544\, km/s$.

The minimum WIMP mass (M_{WIMP}) detectable by direct detection experiments (see Section 2.4.2) can be calculated using eq. 2.16, where A is the atomic mass of the target nucleus and E_{min} is the energy threshold for detecting nuclear recoils. This expression is valid assuming $\nu_{esc} = 544\, km/s$ [1].

$$M_{WIMP}(GeV) \approx \frac{1}{4}\sqrt{E_{min}(keV)A} \tag{2.16}$$

2.4 Detection Methods

If Dark Matter is indeed made of WIMPs, then their interactions with baryonic matter, even though small, should produce some kind of measurable signal, that can potentially be detected by a sensitive enough instrument.

There are two types of detection methods: direct and indirect detection. Indirect detection refers to the observation of WIMP annihilation products (cosmic and gamma rays) in locations where WIMPs have more probability of existing (center of galaxies and galaxy clusters) [24]. Direct detection refers to measuring evidences of the WIMP flux passing through Earth, by detecting scattering interactions between WIMPs and a target located on Earth [25]. Another possible solution to study Dark Matter is to produce it in particle accelerators, like the LHC at CERN [26].

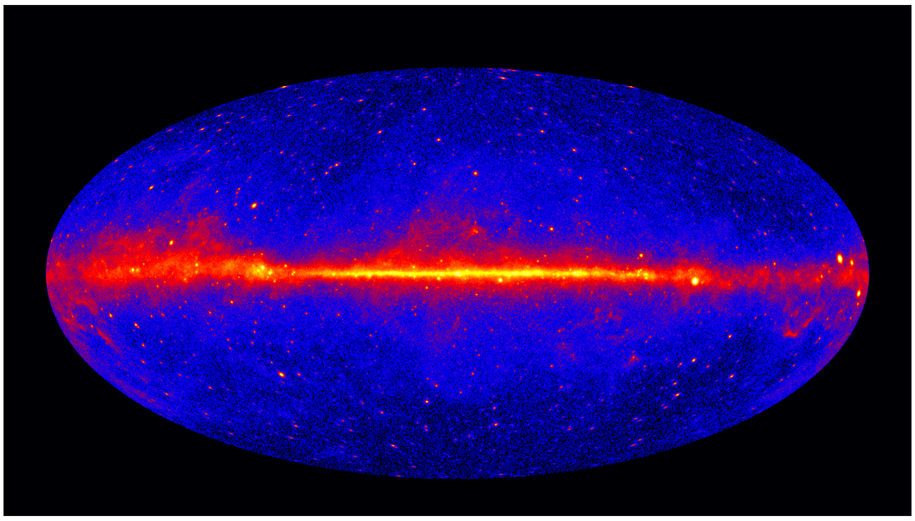


Figure 2.7: Map of the high-energy γ ray emissions in the Universe captured by the FGST. The red and yellow spots are places of higher density of γ rays, and can thus be candidates for the search of WIMP annihilation signatures.

2.4.1 Indirect Detection Methods

From eq. 2.13, the annihilation of WIMPs would result in the production of high energy particle-antiparticle pairs. The resulting pairs could be neutrino/antineutrino or two photons. Since WIMP kinetic energy is small, the energy of the resulting particles will be close to the WIMP mass, so the signature of the observed signal will be a line in the energy spectrum.

One example of an indirect detector for WIMPs is the Fermi Gamma-Ray Space Telescope (FGST), launched in 2008. The FGST measures high-energy γ ray emissions in galaxies that are expected to have relatively large amounts of Dark Matter and only a few known γ sources (e.g. Dwarf Spheroidal Galaxies), with the objective of finding WIMP annihilation signatures. These galaxies are favoured for searching this type of signatures because of their expected high Dark Matter density, something that favours WIMP self-annihilation [5].

The most recent Fermi-LAT results rule out thermal relic velocity-independent cross sections in the 10-100 GeV range. A lower bound for the lifetime of a $\sim$100 GeV dark matter particle was estimated to be $\sim 10^{26}$ s [24].

2.4.2 Direct Detection Methods

The aim of direct detection experiments is to measure the rate of nuclear recoils caused by the scattering interactions between WIMPs and the atoms of a target material. This method of detection allows to obtain a value for the interaction cross section between WIMPs and baryonic matter.

The main challenge of these methods is connected to the expected low event rate, due to the weak nature of the interaction. Another difficulty lies with the energy transferred to the sensitive medium since, as it increases, the expected WIMP interaction rate falls, which reduces the sensitivity of the detection [1]. So, in order to detect these rare events there needs to be great suppression of backgrounds.

Backgrounds can have various origins. The constant flux of cosmic particles that reach the Earth is one of the major sources of background. The signals from its interactions with the target volume would overload the detector with unwanted noise, making the detection of any relevant signal nearly impossible. However, not all backgrounds originate from outside the detector. The target volume and the construction materials of the detector are also sources of background.

To mitigate the effects of these backgrounds a number of techniques are applied. Detector shielding is the primary form of suppression for backgrounds that originate outside of the detector. Its purpose is to filter out the unwanted incoming particles, greatly reducing the amount of signals those particles would generate inside the detector. A common form of shielding is placing the detector underground which has the effect of reducing the flux of cosmic particles. Further shielding of the detector can be achieved by placing it inside a water tank, which filters out γ and neutrons generated underground [1].

Choosing the target material of the experiment is also a crucial step. Materials with high atomic masses are ideal to study spin-independent interactions. This is due to the fact that the cross section of the spin-independent interactions is approximately proportional to the square of the atomic mass. However, lower atomic masses, and lower energy thresholds, are better to study low mass WIMPs. To study spin-dependent interactions, isotopes with non-zero spin are added to the target volume [7]. For example, Xenon is a target material that combines a high atomic mass with the natural presence of non-zero spin isotopes, making it good to study both spin-dependent and independent interactions. Traces of radioactive isotopes are always present in the target volume. Removing all these isotopes is too expensive to be feasible. So, only the isotopes with long half-lives are removed, since these would be a continuous source of background during the entire running time of the experiment.

Common examples of target materials used are: LXe (LZ, LUX, XENON,etc.), LAr (DEAP, DarkSide), NaI (DAMA/LIBRA), among others.

The construction materials of the detector are also sources of background. For the material choice, a particular concern is the amount of radioactive isotopes present (especially ^{238}U, ^{232}Th, ^{40}K and ^{60}Co). The presence of these radioactive isotopes should be reduced to a minimum. In LZ, an intensive screening campaign was conducted to quantify and mitigate the presence of these and other radioactive isotopes for all detector components. The materials chosen for construction were titanium, copper, polytetrafluoroethylene (PTFE) and stainless steel for the external components [1].

Fiducialization is also an effective way of reducing background. This technique consists in dividing the volume of the detector in two parts: an inner (fiducial) volume and an outer volume. The outer volume will "absorb" most of the background interactions, meaning that the greater part of the signals collected in the fiducial volume come from relevant interactions[5]. The signals of the interactions occurring in the outer volume act as an active veto and can also be recorded for future analysis of the background [1].

New forms of background reduction are continuously being studied and developed like, for example, the Outer Detector (OD) used in LZ. This addition is a great asset in reducing backgrounds generated internally as well as external backgrounds, thus acting as an integrated veto system. It also allows to increase the fiducial volume of the main detector because it provides additional passive and active shielding (see Section 2.6) [27].

Despite the effectiveness of the mentioned mitigation methods, they are not "a silver bullet". So, a way to discriminate and reject the backgrounds needs to be put in place. This discrimination and rejection is normally done *a posteriori* as it requires a detailed analysis of calibration runs for both signal and background-like events. This analysis allows to catalog specific signatures (pulse shape, area ratios, etc.) that are from background interactions, and then remove the events that contain those signatures from the final analysis [1].

The validity of the obtained results is deeply linked to the ability of reducing background events, that have the same characteristics as WIMP scattering signals (e.g. neutrons).

When a WIMP scatters of a target nucleus, it deposits energy in the medium. The expended energy can be converted into three possible channels: ionization charge, phonons and scintillation. By using more than one of these options, the rejection rate of backgrounds is increased [25].

Based on the type of readout, direct detectors can be divided into different types

[5]This is partially the reason why liquid noble gas detectors tend to increase in size from one experiment to the other, so that both the fiducial and the outer volume can be increased without compromising the effectiveness of one another.

[25]: Threshold detectors, that use superheated fluid to measure ionization on the target material (e.g. PICO collaboration [28]); Crystal scintillator detectors that measure scintillation of the target (e.g. DAMA/Libra [29]); Liquid noble element detectors, that measure scintillation and ionization of the liquid target (e.g. LZ [1]); Semiconductor detectors, that measure ionization and scintillation in cryogenic crystals (e.g. CDMS experiment [30]); Directional detectors that measure the ionization on a gaseous target to determine the vector direction of nuclear recoils caused by WIMPs (e.g. DRIFT experiment [31])

Since LZ falls into the liquid noble element detectors, only those are going to be approached in the following section.

2.5 Liquid Noble Element Detectors

Liquefied noble gases combine scintillation properties with the ability to let electrons created by ionization drift for long distances [32].

These characteristics make liquefied noble elements very interesting materials to use in Dark Matter experiments. Liquid Xenon (LXe) is the liquefied noble gas that presents the best results for high mass WIMPs. Collaborations such as PandaX [33], LUX [34], XENON [35] and now LZ [1] use this element as their target material. The current best results were obtained by XENON1T with a maximum rejection limit cross section of $4.1 \times 10^{-47} cm^2$ for a WIMP mass of 30 GeV/c^2 [36].

Other experiments like DEAP[6] [37] or ArDM[7] [38] use liquid Argon (LAr) as their target material, but with worse results than their liquid Xenon counterparts.

Next up, the design and working principle of a Time Projection Chamber is going to be presented. Also, the latest results relevant for this work, coming from the LUX and XENON1T experiments, are going to be exposed before giving a more in depth view of the LUX-ZEPLIN experiment.

2.5.1 Time Projection Chambers

A Time Projection Chamber (TPC) is a particle detector used in physics, that allows a 3D reconstruction of the interaction of a particle. It was invented in 1978 by David Nygren, to overcome some of the difficulties present in particle detectors at the time [39]. The original concept is a cylindrical chamber filled with a gaseous

[6]Dark matter Experiment using Argon Pulse-shape discrimination

[7]Argon Dark Matter

detection volume with a multi-wire proportional chamber (MWPC)[8] in each of the endplates. An electrical field is applied to the TPC making the electrons generated by the interactions inside the volume to drift towards one of the MWPC endplates. Each of the MWPCs can give information on the XY coordinates of the primary particle interaction, and the Z coordinate is obtained by measuring the drift time of the electron [40].

Right now, the state-of-the-art for WIMP search is Xenon dual-phase TPCs with PMT arrays serving as endplates. This technology consists in filling the cylindrical chamber with liquefied Xenon (LXe) in equilibrium with its vapor phase (GXe). The LXe fills most of the TPC and is operated at a temperature of approximately -95 $^{\circ}$C at a pressure of $\sim$2 bar (in the case of LZ -97.4 $^{\circ}$C at a pressure ranging from 1.6 to 2.2 bar [1]). A downwards electrical field is applied.

The operating principle is the following. An interaction between a primary particle and the LXe ionizes the Xenon atoms, releasing electrons and generating scintillation light (S1). The applied electrical field causes the electrons generated by the ionization to drift towards the surface of the liquid where they will be extracted by a second, stronger, electrical field to the gaseous phase. The extracted electrons will generate a secondary light signal, which is proportional to the amount of charge created by the original interaction (S2) [41].

The amount of charge that is extracted to the gas phase is affected by electronegative impurities present in the target volume. Due to attachment of the electrons to these impurities, the number of ionization electrons drifting from the interaction position decreases exponentially with time. The time constant of this decrease is called free electron lifetime. The Xenon in LZ will be constantly circulated through a purification system, thus maintaining the impurity concentration low and a free electron lifetime of the order of milliseconds. However, interruptions in the circulation may lead to periods of increased impurity concentration and, consequently, reduced electron lifetime.

Both the scintillation signals are recorded by the PMTs. The signals from the primary scintillation, the one that occurs following the initial interaction between the particle and the LXe, are called S1. They are the first to be detected by the PMTs. The signals originating from the secondary scintillation occurring in the GXe are called S2. Their amplitude is usually much stronger than the S1s and they arrive later at the PMTs, due to the time it takes for the electrons to drift towards the GXe (Fig. 2.8).

[8]type of proportional counter that can give information on the trajectory of a charged particle or photon

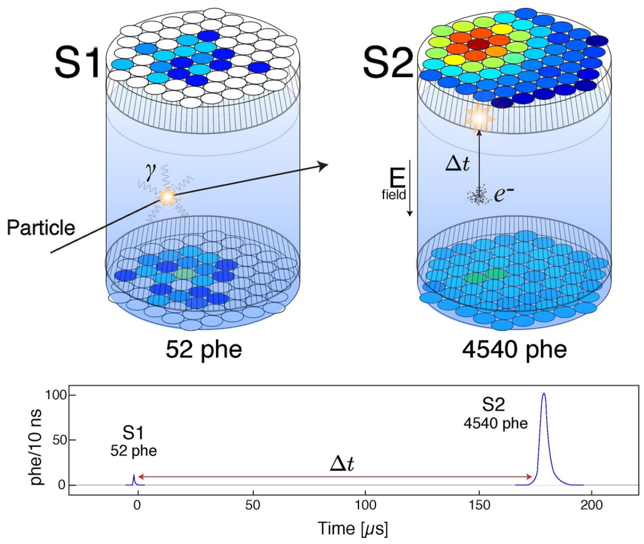


Figure 2.8: Representation of the differences between S1 and S2 signals. The top image shows how S1 and S2 signals are formed inside the TPC. S1 signals have their origin in the light generated by scintillations in the LXe. S2 signals are originated by electrons that drift towards the top of the TPC and produce secondary scintillation in the gas phase. In the bottom image, the different characteristics of the signals can be seen. S2 signals usually produce more phes and are thus stronger [34].

Concerning the comments made in Section 2.4.2 about background mitigation, it is possible to understand why LXe is currently the most used material (and the one who achieves better results) in WIMP search, for example when compared to liquid Argon (LAr). Liquid xenon is a denser material ($\rho_{LXe} = 2.942\, g/cm^3$, $\rho_{LAr} = 1.3954\, g/cm^3$) and an efficient absorber of γs. This means that fiducialization of the target volume is more effective using LXe than with LAr, i.e. it is possible to have a larger fiducial volume while also having a outer volume large enough to significantly reduce the background [41].

A characteristic that is common to all scintillator materials is the different energy loss (dE/dx) for electronic recoils (ER) and for nuclear recoils (NR). It is expected that the signals coming from WIMP interactions are nuclear recoils, with the ER signals generated inside the TPC being from γ or β background events. Having different dE/dx for ER and NR events means that the amplitude ratio between S2 and S1 signals is also different, which leads to a further form of background rejection [41]. However, LXe and LAr differentiate themselves from other scintillators by allowing to measure scintillation and ionization simultaneously, a characteristic that further helps NR/ER discrimination.

Although LXe TPCs are still the state-of-the-art for WIMP detection, better technologies and target volumes are available. One example is solid state detectors using Germanium as a target volume. However, the major limitation of these detectors lies with the feasibility (its both very hard and expensive) of building detectors with a large enough mass to achieve a similar exposure to WIMPs as LXe TPCs [17].

2.5.2 Large Underground Xenon (LUX)

The LUX collaboration was composed by 19 groups from 3 different countries (USA, UK and Portugal). With the intent of improving upon the results from previous dark matter experiments, LUX proposed to build a TPC filled with 300 kg of LXe [34]. The detector was installed 1478 m below the surface at the Sanford Underground Research Facility (SURF) which is located in the former Homestake Gold Mine in South Dakota, USA. As already stated before in section 2.4.2, installing the detector underground is a good form of shielding from cosmic muon backgrounds, since their flux is reduced by a factor of about a million when compared to the surface [42].

By increasing the mass, the exposure is also increased, which means that the large mass of the detector allowed to obtain lower sensitivity limits. The TPC allows to fully reconstruct both the deposited energy and the position of interaction for an event inside the active volume. An inner fiducial volume inside the TPC (100 kg LXe) is established and, within it, the background rate is heavily suppressed due to active shielding provided by the surrounding Xe [34].

The basic design of LUX is similar to that of previous experiments, such as XENON10 and ZEPLIN-II. However, some important advances were included. The most notable ones are the use of titanium with low uranium and thorium contamination as a construction material; the implementation of an high voltage feedthrough outside the shielding; and the use of an 8-m diameter water tank inside which the detector was placed. This last addition served as a shield to gammas and neutrons [34].

Two runs were conducted for this experiment between 2013 and 2016. The first run (WS2013) started on April 21^{st} 2013 and ended on August 8^{th} 2013, which corresponded to a live day count of 95 days. The second run (WS2014-16) started on September 11^{th} 2014 and ended on May 2^{nd} 2016, which corresponded to a live day count of 332 days. The live days registered do not represent the total run time of the experiment, as they are corrected for a number of factors. These factors include periods of detector instability, detector dead time or periods of low electron lifetime [43].

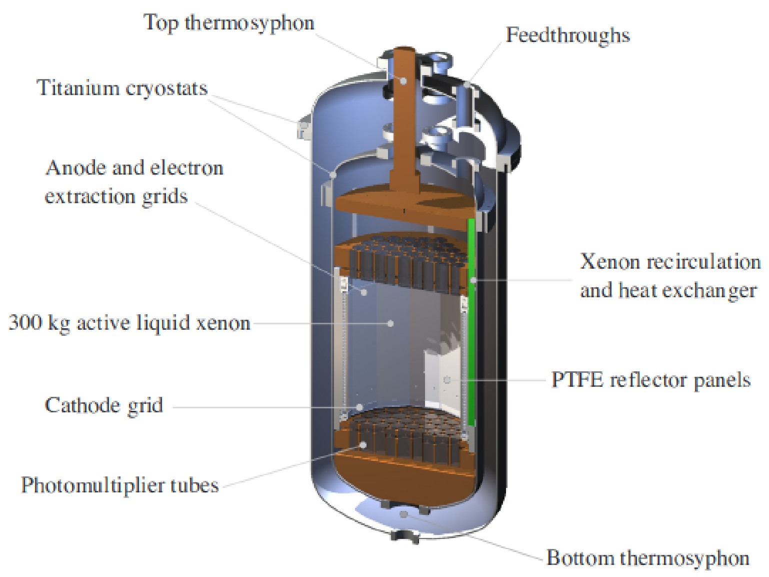


Figure 2.9: Schematic of the Large Underground Xenon detector [34].

A "salting" protocol was put in place during the data collection period, in order to reduce analysis bias. In this protocol artificial WIMP-like events (called "salt") were generated from ^{3}H calibration data and introduced into the data pipeline. On the unblinding stage this allowed to investigate and tune the efficiency of the cuts that were latter applied in the analysis. The analysis of the WIMP signal candidates was performed using a Profile Likelihood Ratio (PLR) statistical test. Nuclear-recoil energy spectra for the signals were derived from a standard Maxwellian velocity distribution with $\nu_0 = 220$ km/s, $\nu_{esc} = 544$ km/s, $\rho_0 = 0.3$ GeV$/cm^3$, average Earth velocity of 245 km/s and a Helm form factor for the target nuclei [43] (see Section 2.3.2).

Regarding the first run, the PLR gave a p value of 0.35 for the background only hypothesis, meaning that the observed events are consistent with the expected background distribution. The 90% confidence limit (C.L) upper limit on the number of expected events ranges from 2.4 to 5.3, for all WIMP masses. For the 90% upper C.L for spin-independent WIMP models a minimum cross section of $7.6 \times 10^{-46} cm^2$ at 33 GeV$/c^2$ WIMP mass [44].

For run number 2, a PLR p value of 0.39 was obtained at 100 GeV$/c^2$, which again agreed with the background only model. The 90% C.L upper limit reached a minimum cross section of $2.2 \times 10^{-46} cm^2$ at 50 GeV$/c^2$, which corresponded to 4.2 expected WIMP signal events. The re-analysis of the combined results of both runs gave an exclusion limit reaching a minimum cross section of $1.1 \times 10^{-46} cm^2$ at 50

GeV/c^2, corresponding to 3.2 events expected [43].

2.5.3 XENON1T

The XENON1T collaboration was made of 23 institutions spread across 10 countries. The experiment was installed at a depth of 1400 m in Laboratori Nazionali del Gran Sasso (LGNS) in Italy. The most notable difference relating to previous experiments is the size of the detector, since it was the first experiment to operate with a mass of LXe over 1 t. The detector was filled with 3.2 t of LXe, with the active target of the TPC being 2 t [35].

A number of improvements were also made to help reducing the background levels. The signals from the PMTs in the water tank were added to the final analysis of the results. This addition allowed to identify muons with a flux of $(3.31 \pm 0.03) \times 10^{-8} cm^{-2} s^{-1}$ and average energy of $\sim 270 GeV$, and muon-induced neutrons, effectively operating as a Cherenkov muon veto [35].

Two experimental runs were conducted. The first run started on November 22^{nd} 2016 and ended on January 18^{th} 2017, recording a total of 34.2 live days. The second run lasted from February 2^{nd} 2017 to February 8^{th} 2018, for a live day count of 278.8 days. "Salted" events were also added to the accumulated data to reduce analysis bias [36].

The parameters used for the data analysis were the same ones for LUX, already presented in Section 2.5.2.

For run number 1, the analysis of the data in the fiducial volume shows an electronic recoil (ER) rate of $(1.92 \pm 0.25) \times 10^{-4}$ events/(kg$\times$ day$\times$ keV$_{ee}$), which is the lowest ever achieved by a Dark Matter experiment. A PLR test was made for the statistical interpretation of the data. The results of the test showed a consistency between the acquired data and the background-only hypothesis. Regarding the spin-independent WIMP-nucleon cross sections, this experiment managed to improve upon the minimum limits, previously established by LUX, for WIMP masses above 10 GeV/c^2. The best exclusion limit achieved was $7.7 \times 10^{-47} cm^2$ for a mass of 35 GeV/c^2 [45].

For the second run, the PLR test returned p-values of p = 0.28, 0.41 and 0.22 for WIMP masses of 6, 50 and 200 GeV/c^2, respectively. These results are again consistent with the background only hypothesis. For the 90% C.L upper limit of the WIMP-nucleon spin-independent cross section a best limit of $4.1 \times 10^{-47} cm^2$ for a WIMP mass of 30 GeV/c^2 was obtained, which is the current best limit achieved [36].

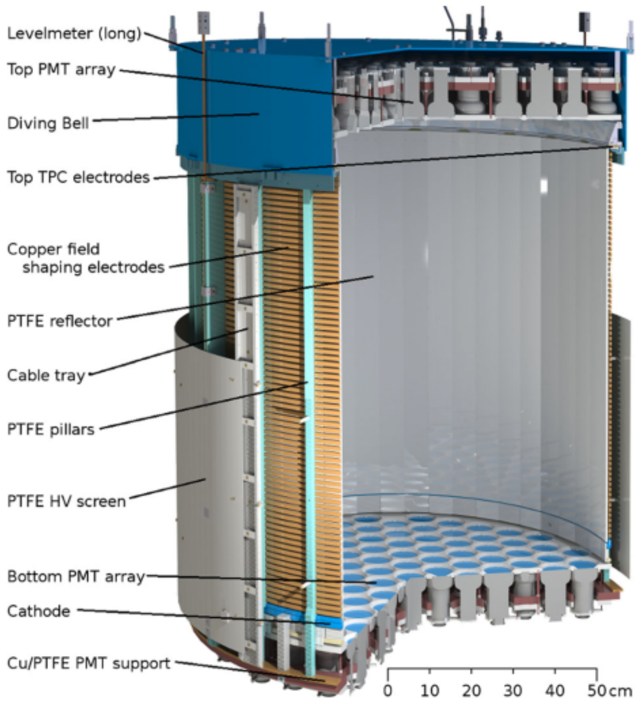


Figure 2.10: Schematic of the XENON1T detector [35].

An upgrade of this experiment, XENONnT, is already under construction. This upgrade is set to increase the active mass of the detector to 5.9 tons, with an expected cross section sensitivity improvement of more than an order of magnitude [46].

2.6 LUX - ZEPLIN (LZ)

LZ is part of the newer generation of experiments aiming at the direct detection of Dark Matter. The technology used by LZ is based on the one used by its predecessor LUX, but with significant improvements.

The detector is now much larger. This enlargement was possible due to the significant improvements made to the background rejection methods, namely by adding the Skin and Outer detectors.

The mass of LXe that fills the TPC is now 10 tonnes, of which 7 tonnes are in the active mass. The fiducial mass is 5.6 tonnes. This will be the largest detector ever built using this type of technology. Like it was already discussed in Section 2.4.2, by increasing the volume of LXe used it is expected that the sensitivity of the experiment also increases, as the extra mass allows to shield and reject more background particles while simultaneously increasing the volume where the relevant

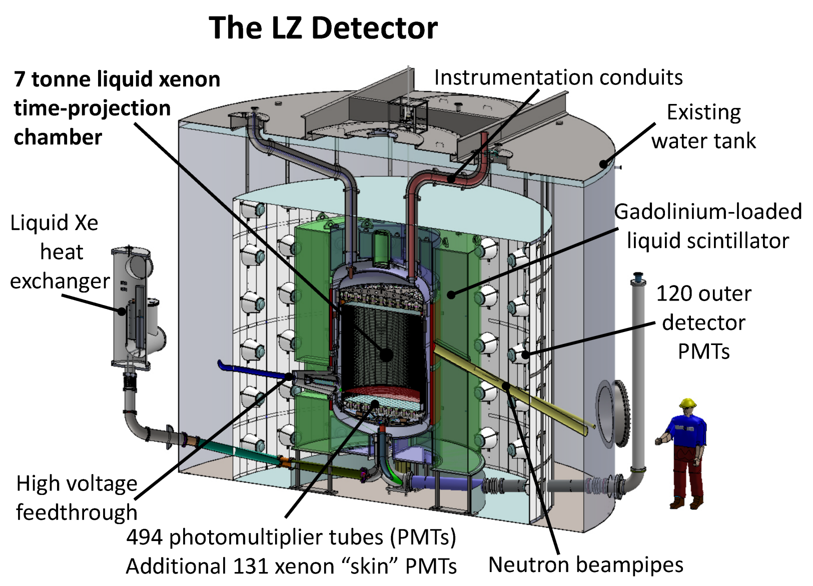


Figure 2.11: Schematic representation of the LZ detector [2].

interactions will occur (fiducial volume).

The LZ experiment will effectively operate with three detectors: the TPC (with two PMT arrays: a top array with 253 PMTs and a bottom array with 241 PMTs), the Skin detector with 131 PMTs and the Outer detector (OD) with 120 PMTs (Fig. 2.11).

The Skin is a novel technique of background rejection used by LZ. It is constituted by a layer of LXe located between the PTFE-clad field cage and the inner cryostat wall, as well as the region underneath the bottom PMT array. The use of this layer will allow to achieve an energy threshold of 100 keV$_{ee}$ for the detection of ER scatters produced by γs or neutrons. In combination with the OD, it will provide a highly efficient method of tagging internal and external backgrounds [2].

The OD is the detector that surrounds the TPC. It will be filled with 17.3 tonnes of gadolinium-loaded liquid scintillator (GdLS). The primary function of this detector is to tag neutron events in the LXe that could mimic WIMP signals, thus operating as an integrated veto system for gammas and neutrons [2], [27]. By adding the OD it is possible to increase the fiducial volume of the TPC.

The combination of the larger LXe mass, the low background construction materials and the new additional detectors, Skin and OD, is what will allow LZ to reach unprecedented sensitivity levels and improve upon the current best results by almost an order of magnitude.

According to the simulations already conducted, this experiment will be able to probe spin-independent WIMP-nucleon cross sections down to $2 \times 10^{-48} cm^2$ at 50

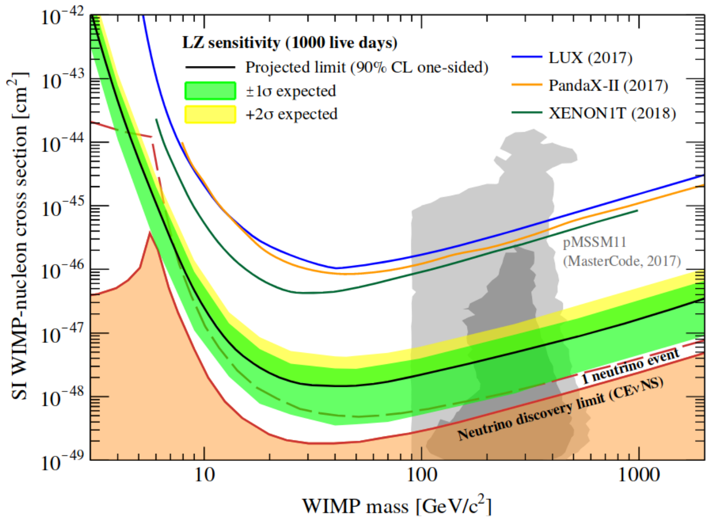


Figure 2.12: Expected sensitivity for cross section measures of spin-independent WIMP-nucleon interactions in LZ (black line). The blue, green and orange lines represent the results of LUX, XENON1T and PandaX-II, respectively, with the result from XENON1T being the current best. The red zone at the bottom represents the zone where neutrino interactions are expected to be dominant [2].

GeV/c^2 over a 1000 live-days long run, which would be the best sensitivity ever achieved so far [1] (Fig. 2.12). This expected sensitivity is very close to an irreducible background called "neutrino floor". Beyond this limit the detected signals would be dominated by solar, atmospheric and diffuse supernova neutrino interactions, and the detection of WIMP signals would become nearly impossible [1].

2.7 Signal Acquisition and Processing in LZ

The PMT signals are shaped and amplified by dual-gain amplifiers, providing High-Gain (HG) and Low-Gain (LG) outputs. The usage of the HG and LG outputs allows to obtain optimal resolution for low and high energy events, respectively, therefore extending the energy range of interactions able to be probed by the detector. Both types of gains are used for TPC and OD PMTs, which allows the latter to detect cosmic muons. The Skin PMT signals are processed by a single gain chain.

The shaping constants used for the amplifier outputs are determined in order to maximize the dynamic range of the acquisition electronics. Various interaction sources are used for the calibration of LZ, with the purpose of characterizing various performance factors such as the NR and ER bands or the response of the detectors [2]. To characterize the dynamic range of the electronics, the energy range of the following sources is used: ^{83m}Kr (32.1 and 9.4 keV transitions), activated Xe (236 keV and 164 keV), and ^{3}H (endpoint at 18.2 keV) [2]. The LG channel uses a semi-gaussian shaping with 30 ns FWTM (Full Width at Tenth Maximum) and a gain of 4, and its dynamic range is defined by the 236 keV Xenon activation line. The HG channel has a gain 10$\times$ higher and a semi-gaussian shaping with 60 ns FWTM. Its response and dynamic range are optimized for single photoelectron (SPE) detection, especially for the krypton and tritium calibrations [2].

The design of the acquisition electronics is largely based on the one previously used in LUX. The PMTs in the TPC will detect both S1 and S2 signals. The Skin detects prompt scintillation signals and the OD detects scintillation signals and Cherenkov radiation. The threshold for signal detection is going to be set at 0.25 phes (phtotoelectrons) for each PMT in the TPC [2].

The data acquisition (DAQ) in LZ is divided into several layers. The signals from the PMTs are continuously being digitized at a rate of 100 MHz and are stored in circular buffers. Waveforms will be collected in Pulse Only Digitization (POD) mode, a mode where only the portion of the digitized signals above a threshold are stored. This is expected to reduce the raw waveform volume by a factor of 50. A header containing information about the POD time position is also stored, to allow the reconstruction of the signal further ahead [2].

Digital filters applied to the acquired data allow the distinction between S1 and S2 signals. For the S1 signals the filter has an integration ranging from 60 to 100 ns. For S2 the filter has a few μs width. By combining these two filters, the trigger processing for the TPC PMTs can be done in three different modes: S1 mode, where only the S1 signals are selected; S2 mode where only the S2 signals are detected; and S1 and S2 mode, where the S1 signals are selected as well as S2 signals that fall inside a defined time range [2].

Once one of these events of interest is detected its data is extracted by the Data Extractor (DE), compressed, stacked and sent to the Data Collectors (DCs) where they are temporarily stored. The Event Builders (EBs) sample the different DC disks, collecting the information that is associated to a specific event. The design of the LZ Data Sparsification system is based on the previously used LUX trigger system. As the DDC-32 digitizers continuously digitize the incoming data, they also process those PODs and extract certain specified quantities like area, height or time of occurrence [2].

These parameters are sent to the Data Sparcifiers (DS), where the PODs are further processed and secondary quantities are generated. The Data Sparsification Master (DSM) analyzes these quantities and decides which waveforms are to be

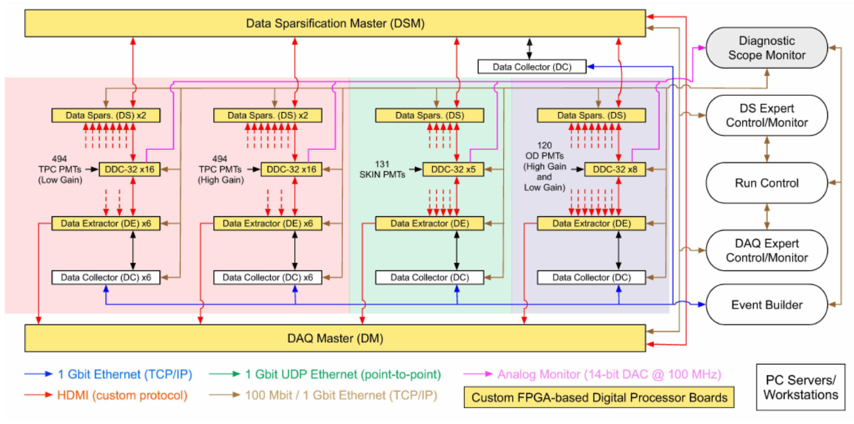


Figure 2.13: Diagram of the data acquisition process in LZ[2].

stored permanently by the DAQ. The quantities that were used to choose the events being stored are also stored alongside the waveform data in the full event files. This enables future cross-checking of data quality and verification of the system performance [2].

To control and monitor the performance of LZ in real-time, an online system is installed onsite. The central point of this system is the Run Control (RC). Connected to it are the DAQ, the EB, the Slow Control (SC) and the Underground Performance Monitor (UPM). These various subsystems are also connected to a database allowing data sharing between all of them. The communications between the RC and the subsystems are done using the Internet Communications Engine (ICE) protocol. The RC controls the type of run being conducted (e.g. calibration, WIMP search), and accordingly sets the type of data acquisition mode used by the DAQ, as well as other required actions from the various subsystems. The Slow Control and the UPM compose the monitoring systems of LZ. The Slow Control has a vast list of tasks that ensure the good functioning of LZ. These tasks include monitoring parameters (temperature, pressures, gas flow, radon concentration in the air, etc.) in and around the detector as well as monitoring the electronics of LZ. The SC also controls the PMT high voltage power supplies, the xenon subsystems (circulation, storage and recovery) and the calibration systems. Regarding the safe operation of the detector, that is ensured by the Slow Control through a system of interlocks implemented in a Programmble Logic Controller (PLC) and an alarm system. The Slow Control GUI (Graphical User Interface) displays the telemetry information collected from the detector sensors, and provides remote control capabilities for authorized operators [2].

The analysis of the acquired LZ data can be divided into two different forms: online and offline. The online analysis is performed by the UPM. It is focused on

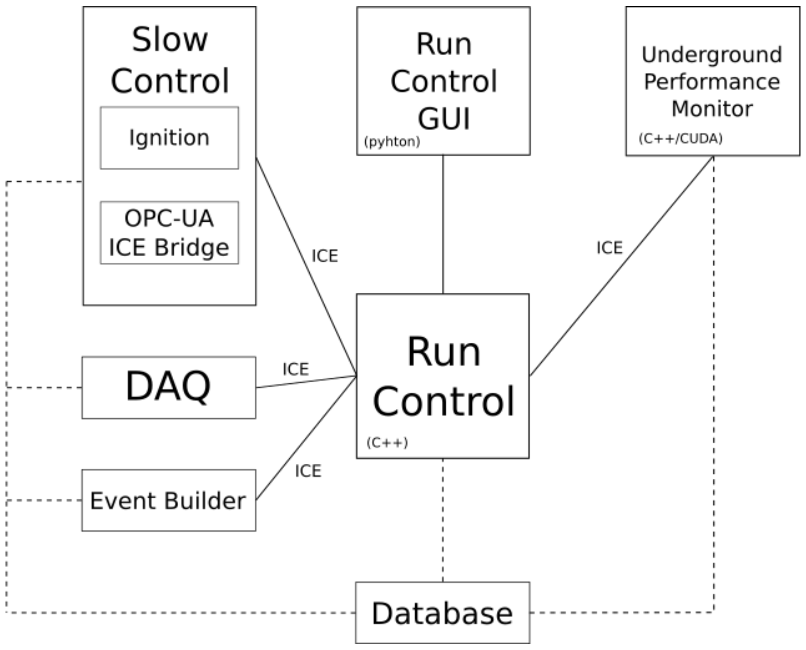


Figure 2.14: Block diagram of the connections between the Run Control and the rest of the online systems.

extracting the necessary quantities needed to characterize both the performance of the detector and the quality of the acquired data. The UPM and its various processing stages are going to be presented more in depth in Chapter 4.

As for the offline analysis it is performed by two Data Centers, one in the USA and another in the UK. Both data centers will run the LZap (LZ analysis package), which will extract the information from the digitized signals. To get that information, the software normalizes the raw waveforms, searches for pulses on those waveforms and classifies the found pulses (e.g S1, S2, single electron, etc.), identifies the type of interaction (e.g. single/multiples scatters, etc.), reconstructs the 3D positions of the events and applies corrections to the data[9]. The analysis process described here is very similar to the one of the UPM, but with a different purpose. In this case the objective is to create what are called reduced quantity (RQ) files. These files are then made available to all the groups in the LZ collaboration for them to perform the physics analysis, and understand the characteristics of the detected events. It is also important to note that, like in LUX, a protocol of ”salting” the data is put in place during the LZ runs, to make sure that the conclusions from the physics analysis are not biased [2].

[9]these corrections are based on Xenon purity, electrical field shape, etc.

3

GPUs

Graphical Processing Units (GPUs) are specialized electronic circuits designed with the purpose of processing and manipulating images. They are used for the acceleration of graphics processing in various devices from mobile phones to computers.

Their structure is engineered to run algorithms that deal with large blocks of data in parallel, something often required in image manipulation. The advantages of these parallelization capabilities lead to an evolution from display-specific devices into general purpose programmable devices, capable of working alongside CPUs in the processing of computationally intensive applications, a technique called GPU-acceleration.

GPU-acceleration consists in combining the best features of CPUs and GPUs in order to improve the performance of general purpose applications. The overall idea is to offload the intensive computational loads, that perform the same operation over large portions of data, to the GPU, and use the CPU for the remaining tasks and coordinating the execution of the program.

3.1 GPU vs CPU

CPUs and GPUs have different design philosophies. As a result their approach to data processing is different. These differences make them complementary to each other.

CPUs are the central point of every computer and, therefore, they need to be able to control a vast set of tasks. Because of this, they are designed to minimize latency, i.e. perform tasks as quickly as possible while also being able to quickly switch between them. GPUs on the other hand are optimized to maximize throughput, i.e. process as many tasks as they can at the same time. These different philosophies are due to differences between their architectures and between their programming models, which in turn makes each of them more suited for different applications.

CPUs have a few cores that can perform more complex tasks and control a larger variety of processes, as they have a broader instruction set. Each CPU thread is optimized to run tasks in a sequential serial way, where tasks are performed one after the other, with the resources being completely focused on a single task at a time.

GPUs on the other hand have hundreds (or even thousands) of small cores and, because of this, are optimized to perform simpler tasks in parallel, where the same calculation needs to be executed over large arrays of the input data. They apply SIMT (Single Instruction Multiple Threads) to achieve its performance advantages. The basis of SIMT is SIMD (Single Instruction Multiple Data), a parallel computing paradigm where the same instruction is applied over multiple data points, introducing data-level parallelism. SIMT takes this further by applying SIMD to multiple threads [47].

A consequence of the difference discussed above is the difference in the amount of operations done per second between the two, measured in FLOPS (FLoating-Point Operations per Second). Currently, GPUs are able to perform about 8 times more FLOPS than state-of-the-art CPUs, with the expectation that this gap continues to grow considering the development speeds of both these technologies (Fig. 3.1).

So, taking into account what was presented, it is observable that GPUs have a large advantage over CPUs when: a) the computation being performed over the data is non-branching and b) the task is computationally intensive.

Having these situations in mind, a careful analysis of an algorithm can uncover performance bottlenecks that can be improved by offloading some of the computational workload to the GPU.

3.2 GPU Architecture

The GPU architecture presented in this section is that of a NVIDIA board, since that is the one being used for this project (Fig. 3.2). The terminology used is also the one used by NVIDIA.

GPU devices can be divided into two main areas: computation and memory. Computation is assigned to the Streaming Multiprocessors (SMs) (Fig. 3.2). Inside them there can be up to hundreds of execution units, called CUDA cores, capable of performing integer or floating-point operations. SMs also possess Load/Store Units (LD/ST), Special Function Units (SFUs) that compute single-precision approxima-

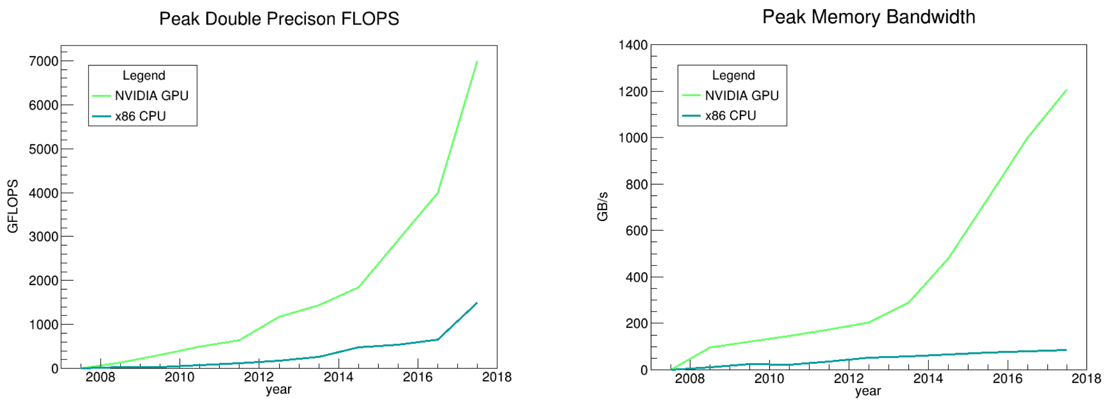


Figure 3.1: Plot comparing the performances of CPUs and GPUs over the last decade. The plot on the left represents the peak double precision FLOPS for CPUs and GPUs, it is possible to observe a large and growing gap between the peak FLOPS of both devices. On the right, the peak memory bandwidth for CPUs and GPUs shows a difference of about one order of magnitude in memory access speed. ©NVIDIA

Memory	Scope	Lifetime	Access Speed
Global memory	Device	Application	Slow
Texture memory	Device	Application	Fast
Constant memory	Device	Application	Slow
Local memory	Thread	Kernel	Slow
Shared memory	Thread Block	Kernel	Fast
Registers	Thread	Kernel	Fast

Table 3.1: Summary of the memory types present on a GPU board, their scope, lifetime and access speed.

tions of various mathematical functions (like log/exp, sin/cos, etc.), and dispatch units and warp schedulers (see Section 3.3) that coordinate the launching of threads [47].

As for the memory, it can be separated into two groups: device-wide and SM-specific memory (Table 3.1). Device-wide memory is visible by all threads in the device, regardless of which SM they are running in. It is the most abundant type of memory on the GPU, but it is also the slowest to access. It is divided into three different kinds: global, texture and constant memory.

Global memory is the largest of them, with its size being up to some GB. It is the only type of memory that is accessible by the PCIe bus that connects host (CPU) and device (GPU). Therefore, data that is going to be copied back to the CPU needs to be stored in global memory before this copy operation occurs. This operation is commanded by the copy engines, which can be 1 or 2, which will, respectively, allow one or multiple memory transfers to be performed at the same time. Texture and constant memories are faster L2 cached memories. However, these memories

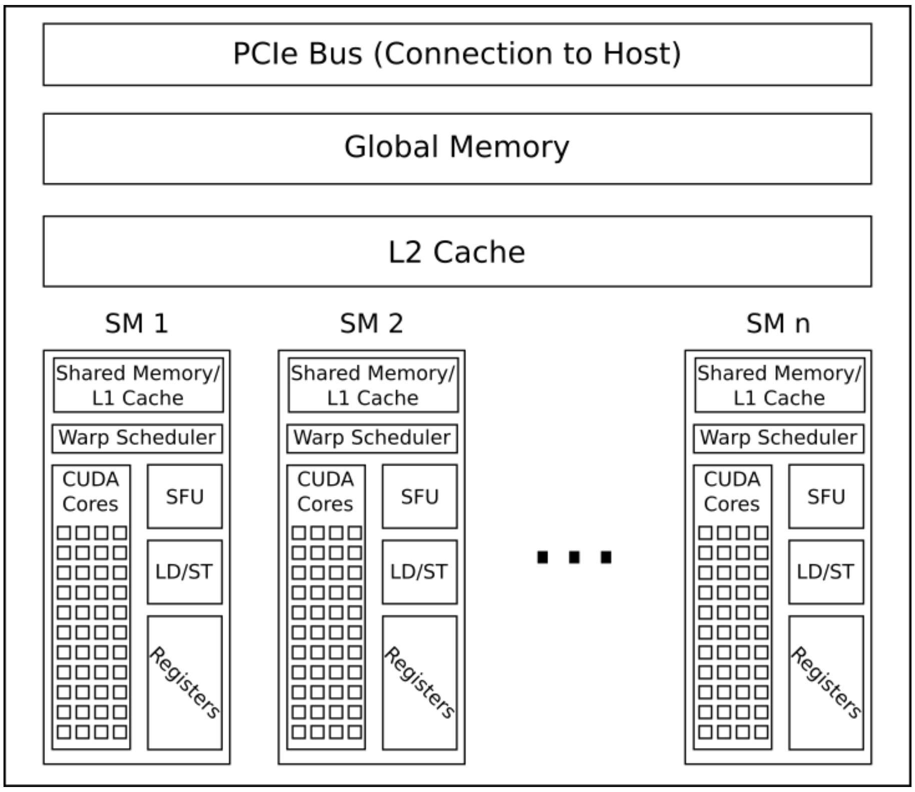


Figure 3.2: Simplified diagram of the architecture of a GPU.

are read-only throughout the entire execution of the application so they should only be used to store variables that are going to remain constant [47].

SM-specific memory is designed for a much faster access. Unlike device-wide memory, it can only be accessed by a specific thread, or by a group of threads residing in the same SM. Registers can only be accessed by the thread that they are assigned to, and are used to store local variables that are being used by that thread [47].

When the data does not fit into the registers, L1 cache is used, and it can be managed by the hardware or by the programmer. When managed by the hardware, it acts as a cached memory that can be accessed by a single thread, and that offers no performance benefits when compared to global memory [47].

When managed by the programmer, is acts as Shared Memory. This is a type of memory that is visible to all the threads inside the same thread block. Performance-wise, it offers the same memory access time benefits as the registers, and is very useful when the same portion of data needs to be accessed multiple times or by multiple threads. However, it needs to be carefully managed, as its size is limited (up to hundreds of kB) [47][48].

3.3 CUDA Programming Model

Compute Unified Device Architecture (CUDA) is a parallel computing platform and API model created by NVIDIA for the programming of GPUs. It is designed to work with programming languages like C, C++ or Fortran.

CUDA applications run in two separate locations, the host (CPU) and the device (GPU). The host is the one that controls the execution of the program. It dictates which operations are offloaded to the GPU, when they are offloaded, and takes care of the data transfers

Communications between host and device are then of the utmost importance. These communications can be simple function calls or more time consuming memory transfers. Memory transfers is the name given to the interchange of data between host and device, and vice-versa. Since host and device are effectively separate locations, they cannot access each others memory directly. The data to be processed on the GPU must be loaded from disk to physical memory (RAM), and then transferred from there to GPU global memory (see Section 3.3.1). Once there it becomes available to all the SMs. Likewise, when data processing is complete on the GPU, the reverse path needs to be followed to update the data on the host side [49]. Because memory transfers are limited by the bandwidth of the PCIe bus that connects host to device, they can be a limiting factor in algorithm performance.

The code designed to run specifically on the device is encapsulated inside what is called a kernel. A kernel contains the code which will be executed by every single thread. The execution of different tasks inside a single thread is sequential. During kernel execution, threads are launched in sets of 32, called warps. A group of warps is called a block. The threads inside a block can all share memory between them using Shared Memory. To optimize the performance, the number of threads inside a block should be a multiple of 32, to ensure that all threads in a warp belong to the same block [47]. Finally, the set of blocks that are launched inside each SM is called a grid. Both grids and blocks can be 1D, 2D or 3D. This allows to apply different data accessing strategies for the cases when the dataset is one-, two- or three-dimensional.

Another important thing to note in host-device relations is that, unless told otherwise, CPU will not wait for the completion of a GPU task, and will continue to execute its own assigned tasks. Synchronization between host and device can be either forced by the programmer or by operations that force/require synchronization between the host and the device. Memory transfers (*cudaMemcpy*) are one of those operations. As data is being moved around between host and device, syn-

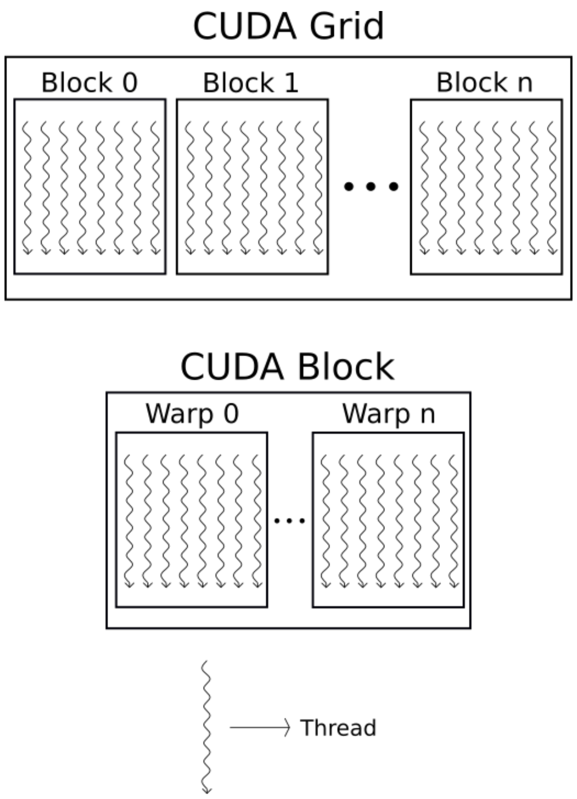


Figure 3.3: Diagram showing the relations between CUDA grids, blocks, warps and threads.

chronization between them is needed to ensure that the proper data addresses are updated. Depending on the type of memory used by the host, pageable or pinned (see Section 3.3.1 for more information), the memory transfer operation might enforce host/device synchronization (pageable memory) or synchronization might only be enforced by the programmer (pinned memory).

Memory allocation in GPU global memory (*cudaMalloc*) is an intrinsically blocking operation on the GPU, meaning that the GPU cannot perform any other operation while memory is being allocated. This blocking nature forces synchronization across all threads in a device. Knowing when to and when not to synchronize host and device is important to extract the maximum performance from the algorithm, while also ensuring that the data is being correctly processed.

Another important characteristic of CUDA is that SMs can execute tasks independently from each other. For the programmer to explicitly control where kernels are being executed, CUDA streams can be used. Each CUDA stream is assigned to a streaming multiprocessor. This allows to run different kernels concurrently in various SMs, or to spread the work of a kernel by more SMs, by specifying which SM is to be used for the kernel launch. The use of CUDA streams can also give space to a significant performance boost regarding memory transfers (see Section 3.3.1).

With CUDA streams, *cudaMemcpy* operations can be performed asynchronously, i.e, one SM can be copying data from/to the host, while another SM is executing a kernel [50].

The general workflow of a CUDA program is quite straightforward and simple to understand. Firstly, memory needs to be allocated on the GPU board and then the data is transferred from host to device (HtoD). Once on the device, it becomes available for processing by the kernels. When the kernels have finished processing, data is sent back to the host (DtoH) and memory is deallocated on the device, finishing the program.

However, even though designing a CUDA application can be quite simple, optimizing it in order to fully utilize the GPU parallelization resources and obtain a significant performance gain, is a lot more tricky. The first place to look is into memory transfers. As the size of the data to be transferred increases, the time necessary for their transfer also increases. Asynchronous transfers can give good performance boosts, but their use might not be possible, or viable, depending on the GPU hardware used. Another good way to improve memory transfer is to batch various smaller transfers into a larger one, avoiding the overhead that comes with starting a new copy operation.

For the optimization of the kernel there are many factors that should be considered. One is to avoid thread divergence. This is a phenomenon that happens when conditional jumps (for loops, if/else statements, etc.) are implemented into the code. The jumps will split the threads, making some follow a certain code path while others will follow a different one. Performance of the kernel is severely deteriorated by thread divergence, because what effectively happens is that while some threads are executing an instruction, the others will be kept idle waiting for their turn. When this happens, the execution of the relevant portion of code turns from parallel to serial.

Memory access is also a common bottleneck when it comes to kernel performance. GPU global memory accesses are the slowest of the types of memory present on the GPU so, a good practice, is to reduce the total amount of reads/writes into it. Using local variables comes in very handy for these situations. As they are stored in the registers, their access is about $100\times$ faster than global memory. So, data can be read only once from global memory into a local variable, computations would be performed on that variable, and in the end writing to global memory would only happen once.

Unfortunately, not every memory access problem can be solved by local variables. Structures that occupy larger amounts of memory, like arrays, cannot be stored in registers. For these cases shared memory can be used. This type of memory is visible to all the threads inside a block, and is very effective in cases where the same data needs to be accessed multiple times by different threads. Synchronization between threads is needed when shared memory is used, to ensure that a thread does not try to access memory that as not yet been loaded. A special level of care is necessary when using shared memory in divergent code. In these cases it is necessary to guarantee that all threads can reach the point where they will be synchronized, or else some threads might be left hanging for an indefinite amount of time waiting for synchronization [48].

3.3.1 Pinned vs Pageable memory

It was already mentioned that the type of memory used by the host can have a significant impact on the speed of memory transfer operations and, therefore, on the overall performance of the algorithm. This section intends to give a better understanding of why that impact occurs, and how pinned memory can be used as an advantageous method to improve the performance of an application.

When memory is declared as pageable, the host can move it in or out of physical memory according to its current needs. If the memory is being accessed by some task, it stays in physical memory. If not, it is moved to disk and the space it previously occupied in physical memory is freed for other applications to use.

When memory is declared as pinned (or locked), it always stays in physical memory, until it is deallocated. Pinning memory has the disadvantage of reducing the total amount of physical memory available to the OS, which can have a negative effect on the overall performance of the host.

As was stated before, GPUs do not have direct access to host memory, or better, GPUs do not have direct access to the host disk memory. When they need to access data from there, the CPU needs to firstly load it from disk into physical memory (RAM), and then the GPU transfers the data from physical memory to its own global memory (Fig. 3.4) [49]. By allocating the memory as pinned, the extra step of loading the memory from disk to RAM is removed (Fig. 3.4). This allows to asynchronize memory transfers with kernel launches, which can reduce a lot the running time of the algorithm.

In CUDA, there are two ways of allocating pinned memory. One is using *cudaMallocHost* to allocate a block of memory as non-pageable, and then copying from pageable memory into this new block. This method has the disadvantage of using more memory than what is needed, since it creates a duplicate of the original mem-

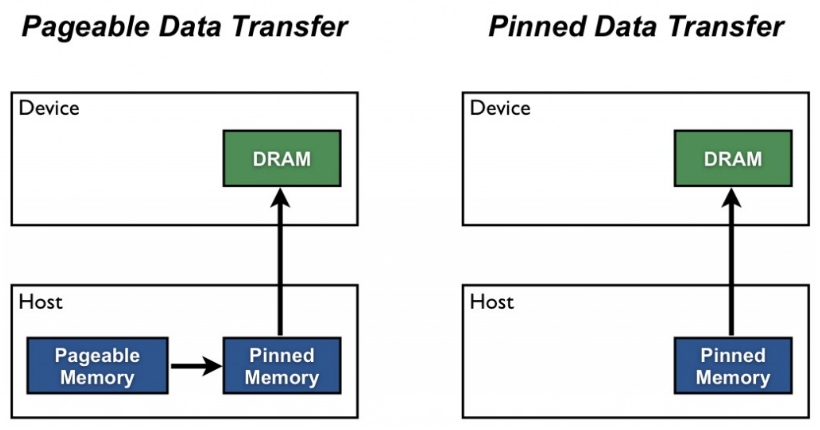


Figure 3.4: Diagram showing the differences between memory transfers using pinned and pageable memory [49].

ory block. It also adds two extra copy[1] operations which also adds more time to the memory transfer.

The other method is using *cudaHostRegister* which registers an already existing block of memory as pinned without needing to allocate a new one. Compared to the previous method it has the advantage of not creating duplicates in the memory. A memory block pinned by this method can be returned back to pageable memory simply by unregistering it. However, only GPU boards with CUDA capability higher than 1.1 are capable of using this option.

The use of pinned memory can be combined with the use of CUDA streams to asynchronize memory transfers with kernel executions. This is performed in the following way: each CUDA stream is assigned to copy, process and copy back a portion of the data; inside each stream the tasks remain serial but across different streams overlaps may occur. For example, while stream n is copying its portion of the data into the GPU, stream $n + 1$ can be executing its kernel, while stream $n + 2$ is copying its data back to the host (see Fig. 4.8 in Section 4.2.2 for a visual representation of what was described) [50].

3.4 GPUs in Physics

This section intends to give a quick overview of how GPUs are currently being used in different areas of physics, and show some of the performance boosts that were obtained.

[1]one to copy the data from pageable to pinned memory, and another to copy the data the other way round

3.4.1 Medical Physics

Medical physics is an area that relies heavily on computationally complex algorithms, used for image processing and reconstruction. Ideally, these algorithms have to be processed as fast as possible to speed up the diagnosis process, as well as allowing the technician to alter parameters interactively and check the results while the patient is still being tested.

Typically, the algorithms used in medical imaging consist on inverting a linear transformation, which can be mathematically described by a matrix multiplication [51] and to small operations on the pixel level [52], two areas where GPUs are very efficient.

The filtered backprojection (FBP) algorithm was the first successful implementation of one of these algorithms on a GPU [53]. It is divided in three steps: 1) multiplication of each projection by a geometry determined factor, 2) convolution of the projection with a ramp filter and 3) backprojection of the filtered projections. While the first step is always ported to GPU, different applications have implemented the entire algorithm in GPU [54], while others have implemented steps 1) and 3) and used CPUs for the filtering step [55].

Iterative reconstruction algorithms are also used, and provide better image quality while exposing the patient to a smaller amount of radiation. Alongside being computationally expensive, they are also sequential processes by nature, thus making them difficult to implement on GPU. The key to their acceleration is to use the GPU to speed up the mapping between the image and projection domains. One example is the Total Variation (TV) method, whose GPU implementation reduced the reconstruction time from hours to minutes, also enabling the realization of studies that require a large number of Cone Beam Computer Tomography (CBCT) reconstructions, such as characterizing the relation between image quality and radiation dose applied [52].

Dose calculation algorithms based on Monte Carlo simulations have also been implemented. Their statistical and random nature does not bode well with the GPU programming model, as they are natural sources of thread divergence. Early implementations assigned a thread to track the history of an original particle and its secondary particles, achieving speed ups of about $5 \sim 6.6\times$ [56]. Further developments divided particles into different arrays based on their type, which helped reduce thread divergence. The speed up achieved increased to $69.1 \sim 87.2\times$[57]

3.4.2 Particle Physics

Particle physics relies heavily on simulation and analysis algorithms. Both these activities require large amounts of computational power. Due to that achieving good temporal performance is expensive using CPUs only. Therefore, GPUs allow for an avenue of improvement of already existing algorithms and to expand the reach of simulation or analysis systems, enabling new studies to be performed.

An algorithm to evaluate the radiation spectra from relativistic charged particles traversing oriented single crystals has been very recently implemented on a CPU/GPU combination. The problem is, in its essence, the computation of a radiation integral. The implementation was divided in two parts: calculation of the trajectory and calculation of the integral. Trajectories are evaluated on the CPU since the number of necessary operations is not enough to produce performance gains by passing it to GPU. The radiation integral is computed on the GPU where each thread is assigned to one integral. This separation of tasks allows trajectory and integral calculation to run simultaneously. The algorithm was tested by measuring the computation time of a certain amount of radiation integrals. A speed up of 205.1× was reported for high-end GPUs when compared to high-end CPUs [58].

Another implementation was the simulation of optical photons. By profiling the Daya Bay Geant4 simulation of its neutrino detector it was shown that the propagation of optical photons consumed 95% of the CPU time. Due to the characteristics of these detectors an external simulation for optical photons can be included in the Geant4 simulation of the other particles. So, the optical photon simulation was passed to GPUs. The only parameters copied to GPU memory are the number of photons in order to generate and a line segment along which to generate them, this is done to reduce transfer overheads. Then, CUDA libraries are used to perform the simulation. A speed up factor of 200× was reported for this implementation [59].

3.5 Real-time applications

Applications that need to perform real-time operations require an excellent temporal performance in order to process the incoming data as quickly as possible. Here GPUs gain another level of importance, especially in the situations where it is not possible to install large amounts of CPUs.

One case that is becoming prominent is that of self-driving cars. Eliminating the human factor means that the car needs to automatically scan for information on the road, such as other vehicles, pedestrians or traffic signs. Regarding the latter, implementations for detecting and recognizing traffic signs fall short in three main aspects: 1) invariance to illumination, as the method relies on analyzing images acquired by a camera, variations of illumination can also lead to errors in sign recognition; 2) difficulty in detecting diverse traffic signs, something that requires the usage of multiple detectors; and most importantly 3) difficulties regarding processing time.

The solution proposed in [60] divides the input image into several blocks and then simultaneously performs the classification of each block. They refer to this technique as parallel window searching. Each of the windows is then divided into 4 parts and an histogram of its features is created for each of them. The 4 histograms are joined together and a deep-learning algorithm is applied to classify the signal based on the extracted features. The method was compared to one of the highest ranked methods currently developed, the Aggregated Channel Feature (ACF). Regarding accuracy of detection the authors report an improvement ranging from 5 to 15% for different daylight datasets. For nighttime datasets it is reported an accuracy of 89.5%, which results in an improvement of about 74%, significantly improving on problem 1). Regarding time performance, the GPU algorithm presents a speed up of about 144× when compared to the CPU ones [60].

Still on the subject of object detection, a real-time implementation of the histogram of oriented gradients descriptor (HOG) for pedestrian detection from images, has also been implemented on GPU. Like the traffic sign recognition algorithm it uses a sliding window method to scan the image, which means that a window is moved across all of the image in order to compute the HOG descriptor. The authors report to have achieved the same detection performance as the CPU version of the algorithm, with a temporal performance increase of 67× for colored images and 95× for grayscale ones [61].

Synchrotron experiments also benefit from real-time monitoring. The state-of-the-art detectors used can acquire images at a rate of 100 fps, producing large quantities of data. This data can be transferred in real-time to the processing site, but real-time processing was not yet achieved for CPU applications. This is a desired characteristic for synchrotron experiments, because the time slots at which they can be performed are limited. So, if an error occurs the experiment can only be repeated when a new time slot is available. With real-time monitoring this problem is avoided because, when a problem is detected, adjustments can be made immediately. In [62] a platform-independent framework for real-time monitoring of synchrotron experiments using CPUs and GPUs is proposed. The framework performs a filtered back-projection (FBP) on the acquired images, where every step of the FBP can be processed by the CPU, by the GPU or by both. The reported performance increase is of 28× for a single slice and of 7× for a full volume reconstruction [62].

Many more examples could be presented [63] [64] [65] [66], but from the ones in this section a pattern starts to arise: most of the real-time applications using GPUs are related to image processing. However, they can still be considered relevant for this work due to two main reasons: 1) the basic principles behind it are the same - analysing a signal and extracting the relevant features from it; and 2) they show the potential of using GPUs in real-time applications that are computationally intensive.

4

Underground Performance Monitor

The Underground Performance Monitor is a LZ subsystem installed onsite. Its purpose is to monitor in real-time the performances of the TPC, Skin and OD detectors. To achieve this, it applies a performance oriented analysis to the raw data output of the detectors. The results of this analysis are then used to assess the performance parameters of the detectors and, if deemed necessary, trigger an alarm.

Previously, in Chapter 2, the relations between the online systems, from the point of view of the Run Control, were explained. Fig. 4.1 shows the same relations, but now from the perspective of the UPM. The UPM communicates with the Run Control (RC), the Slow Control (SC), the Event Builders (EB), and the state and conditions database. The RC tells the UPM when the EB has events ready to be processed. It also indicates which of the configuration files stored in the database is to be loaded to execute a specific analysis. If an abnormality in performance is detected, the UPM signals the SC which will in turn react by triggering the necessary alarms.

In this scheme of relations, the UPM acts as an ICE server, with the RC being an ICE client. Communications between UPM and SC are performed using MODBUS, a protocol widely used in industrial monitoring and control applications. Communications with the database are done using SQL.

The UPM framework is divided into 5 different asynchronous services (docks): File Service, Input, Analysis, GPU and Output docks (Fig. 4.2). These areas communicate between them using asynchronous workers (bots).

The File Service is the entry point of the UPM. When a file in the EB is ready to be processed, the RC tells the UPM to go fetch that particular file. The files will then be made available for the bots to load events at the Input dock.

When at the Input dock, the file is placed in a queue waiting to be processed. Once an event has been loaded by the bot, it is moved to the Analysis dock where

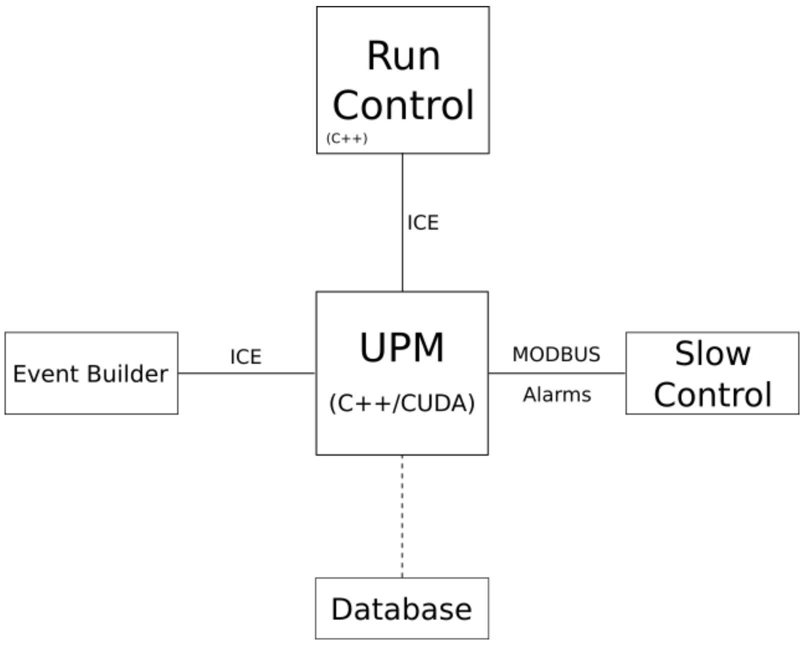


Figure 4.1: Block diagram of the connections between the UPM and the other LZ subsystems. The connections between the remaining LZ subsystems have been purposefully omitted.

the analysis is performed.

In the Analysis dock there is no waiting queue and the execution is asynchronous. This means that N events can be analysed simultaneously, N being the number of CPU threads used. Once at this dock, the events are processed to extract the quantities necessary for the monitoring of the LZ detectors. The signal processing chain (see Section 4.1), allows its algorithms to run either on the CPU or the GPU, based on event-based criteria (e.g. size of the event, origin of the data, etc.). If the analysis is to be performed on the CPU, the bot stays at the Analysis dock. If the analysis is to be performed on the GPU, the bot is moved to the GPU dock.

Inside the GPU dock the bot is placed in another queue, waiting for a GPU to be available. After completing the execution on the GPU, the bot returns to the Analysis dock for the next step of the chain, where the described process is repeated.

When the analysis is finished, the bot is sent to the Output dock where data becomes available to be plotted using an HTTP viewer or sent to the SC to trigger any alarms, if an anomaly is detected. The HTTP viewer and the UPM GUI serve as the interface with the human operator. It allows them to visualize the results of the analysis performed by the UPM.

As of the beginning of this project, the framework of the UPM was already fully developed for CPUs, using the programming language C++.

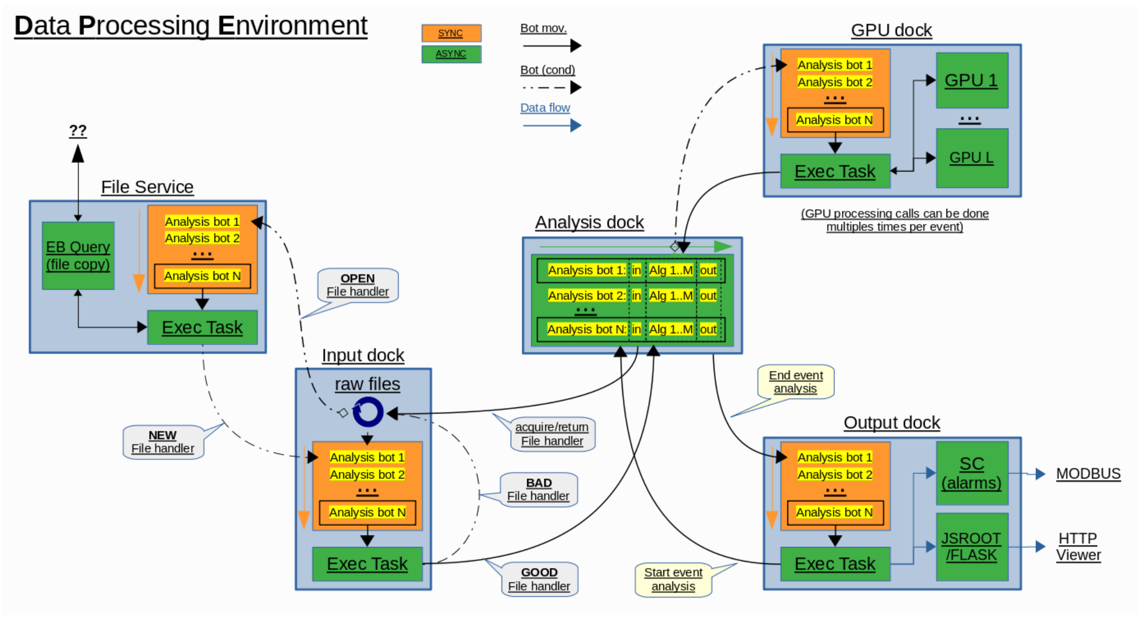


Figure 4.2: Flowchart of the UPM framework.

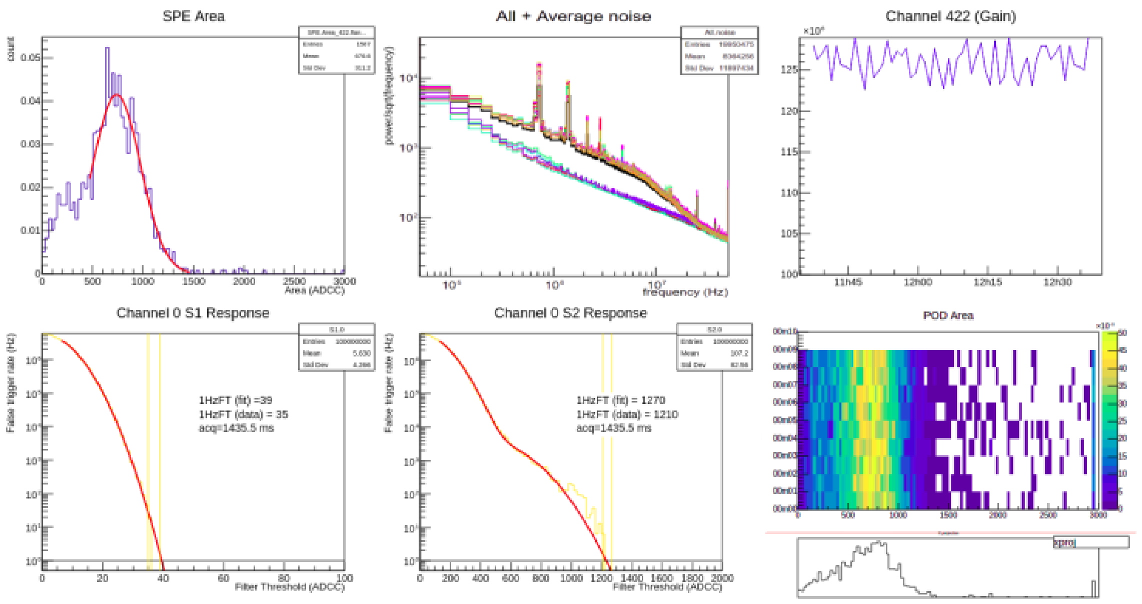


Figure 4.3: Examples of some plots produced by the UPM HTTP viewer. The viewer serves as a human interface so that the operators can visualize important parameters of the detector performance. These parameters can be the response of each channel to S1 and S2 signals, the areas of the PODs, variations on the gain, etc. The evolution of these parameters with time can also be monitored. One plot that has been particularly useful is the one characterizing the noise in each channel (top middle image). A lot of unwanted frequencies are observed in some channels, which will have a detrimental effect in detection performance.

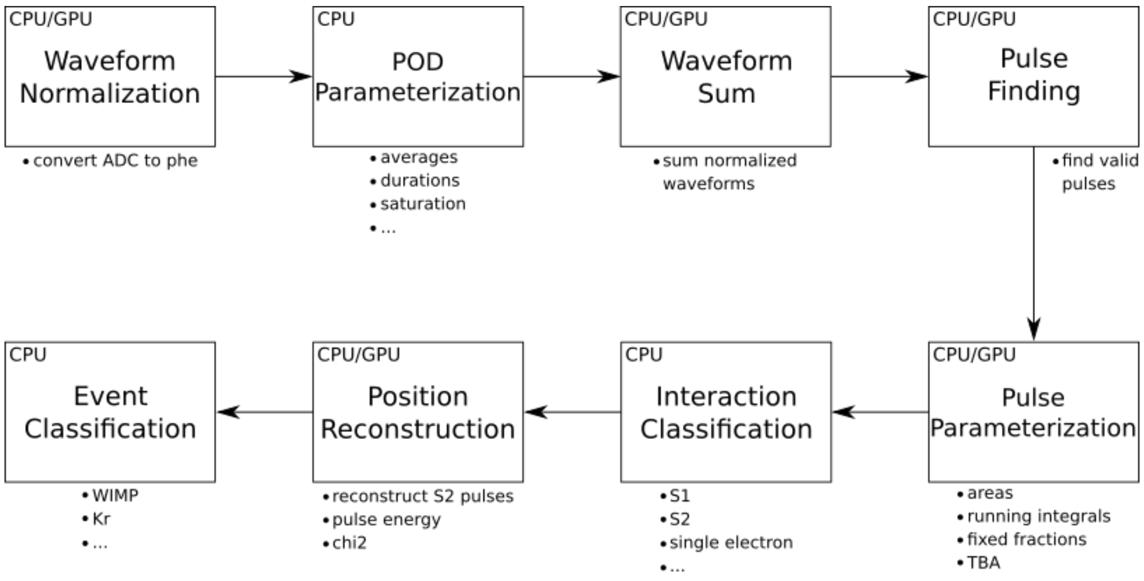


Figure 4.4: Diagram showing the steps of the Signal Processing Chain.

4.1 Signal Processing Chain

The purpose of the signal processing chain is to produce a set of reduced quantities which can be used to assess the data being acquired and ensure its quality. It is composed of various algorithms: waveform normalization, POD parameterization, waveform sum, pulse finding, pulse parameterization, interaction classification, position reconstruction and event classification (Fig. 4.4). Other analysis chains are also available to perform more specialized analysis such as analysis of noise or of the DAQ trigger efficiency.

Some steps of the signal processing chain can be processed either on the CPU or on the GPU, depending on a number of conditions, event size or GPU occupation for example. These steps are: waveform normalization, waveform sum, pulse finding, pulse parameterization and position reconstruction.

The input of the chain is an array containing the raw data. The data in the array is grouped by PODs. Each POD contains the data from the digitized pulse and a timestamp that allows the reconstruction of the timeline of the event. The PODs are then ordered by channel ID, so that the PODs corresponding to the same PMT array and the same detector are stored contiguously. Table 4.1 summarizes the division of the data in PMT arrays and detectors, and indicates the channel IDs corresponding to each of them.

The first step of the chain is waveform normalization. Here the values of the channel ADCs stored in the raw data array are converted into photoelectrons (phe).

PMT Array	**Channel IDs**	**Detector**
TPC Top HG	0 - 253	TPC HG
TPC Bottom HG	300 - 541	
TPC Top LG	1000 - 1253	TPC LG
TPC Bottom LG	1300 - 1541	
Skin Top	600 - 693	Skin
Skin Bottom	700 - 720	
Skin Dome	730 - 748	
–	800 - 920	OD HG
–	1800 - 1920	OD LG

Table 4.1: Detectors, PMT arrays and respective channel IDs.

The conversion is done using eq.(4.1)

$$n_i = g \times (z - r_i) \tag{4.1}$$

where n_i is the normalized data, r_i the raw data, g is the PMT channel gain, calculated for each channel[1], and z is the POD baseline, calculated for each POD.

Still regarding the PODs, they are parameterized in order to extract quantities such as averages, duration, amplitudes and regions where the ADCs are saturated.

The analysis then advances to the next step of the chain: waveform sum. Here, the normalized waveforms from each channel are summed together for all detectors. The summed waveforms are then made available in two formats: a POD format, and a continuous padded format. In the latter the gaps between the PODs are padded with zeros, in order to obtain the real signal from the event and to allow them to be used directly by the FFT filters used in the pulse finder stage.

Next, the pulse finder is called for all the summed waveforms. Its purpose is to delimit the regions where there are valid pulses. They later will be classified as S1, S2, single photoelectron (SPE), etc., based on their characteristics.

The pulse parameterizer is then called to calculate specific quantities (areas, fixed time and area fractions, etc.) from the pulses. The waveforms are classified, by the interaction classifier, based on the type of interaction that originated them.

The final step of the signal processing chain is position reconstruction. In this step, the data from the pulse parameterizer is used to select the strongest S2 pulses and reconstruct their position in the TPC.

Apart from the position reconstruction, which only uses the data from the TPC Top PMT array, all other algorithms use data from all the detectors.

In the following sections, the implementation of all these algorithms in the GPU

[1]these values are obtained during LZ calibration runs

dock is going to be discussed in depth.

4.2 Waveform Normalization

4.2.1 Overview

Waveform normalization is the first step in the analysis of an event. It takes the raw data array and applies the same linear equation (eq. 4.1) to all the elements of the array, converting them from ADC values to phe. The normalized data replaces the raw data in the same array, optimizing memory management.

Two major factors were taken into consideration when deciding the implementation of this algorithm: the size of the array and memory transfer time.

It was known beforehand that the number of elements in the array is, generally, greater than the maximum number of concurrent threads running on the GPU. So, to work around this, a decision regarding the way each thread accesses the raw data had to be made. Two solutions were considered.

The first solution (WN1) was to evenly distribute the number of elements in the array across all used threads. This way, all threads would have the same computational workload. WN1 has the advantage of always using the maximum number of threads possible, which is particularly efficient for small events. However, due to the PODs having variable sizes, there would eventually be a need for a thread to process data from different PODs. To do this, the values of gain and zero need to be updated, which increases the total number of accesses to GPU global memory. This factor becomes a major limitation of this approach as the size of the events increase.

The second solution (WN2) was to assign a single POD to each launched thread. Here, the computational workload is not as evenly distributed across the threads as in WN1. The number of launched threads is also not fixed, as it varies with the number of PODs. In the cases where the number of PODs exceeds the maximum number of concurrent threads, there needs to be a scheduling between the launch of new thread blocks and the end of the execution of the existing ones. Fortunately, in this case, this proves to be an advantage, as it allows the GPU to be working closer to its full potential during a longer period of time. Another advantage is that there is no need to update the values of the gain (g) and zero (z), which greatly reduces the number of accesses to global memory.

In the end, WN2 was chosen, due to the simplicity of its implementation, combined with the fact that it presents a performance increase of about 5 to 10× for larger events (see Fig. 5.1b in Section 5.3).

Regarding the memory transfer time, the objective is to reduce it as much as possible. With this in mind, only the strictly necessary information was transferred between host and GPU. Before launching the kernel, the raw data array and a structure containing POD information are copied from the host to the GPU. After kernel execution, only the normalized data is copied back to the host, as the POD information remains unchanged.

A further layer of memory transfer optimization can be put in place by using asynchronous memory transfers. This allows to perform simultaneously memory transfers and kernel executions, therefore reducing the total run time of the algorithm by about 10 ms for the larger events (see Fig. 5.3b in Section 5.3). However, the method chosen for the asynchronous solution is not available in all GPU architectures (see section 3.3.1). So, the final version of the code contains an option to use synchronous or asynchronous memory transfers.

4.2.2 Workflow

The algorithm is divided into two parts: data pre-processing and waveform normalization. The pre-processing is performed on the CPU because the access and manipulation of the stored raw data is easier and faster. It is performed before sending the algorithm to the GPU queue, to not wast queue time. Here information about the PODs is collected, calculated and stored in a structure. This structure contains an indication of the channel corresponding to the POD, channel gain (g), the start index of the POD in the raw data array (POD offset), the absolute index of the start of the POD in the general timeline (POD idxStart), POD size and POD zero (z). The POD zero is calculated based on the samples directly before and after the POD trigger, causing z to vary from POD to POD. Note that g is the same for all PODs in the same channel.

After finishing pre-processing, the algorithm is sent to the GPU queue and the waveform normalization step begins. Here a pointer to the buffer that contains the raw data is passed to the function and the buffer is processed in place, i.e. the normalized data replaces the raw data inside the same array. This choice optimizes memory usage inside the UPM. The pointer to the POD information structure is also passed to the function.

By default, the maximum number[2] of CUDA streams is created, but only the

[2]this maximum number is equal to the number of SMs in a GPU board

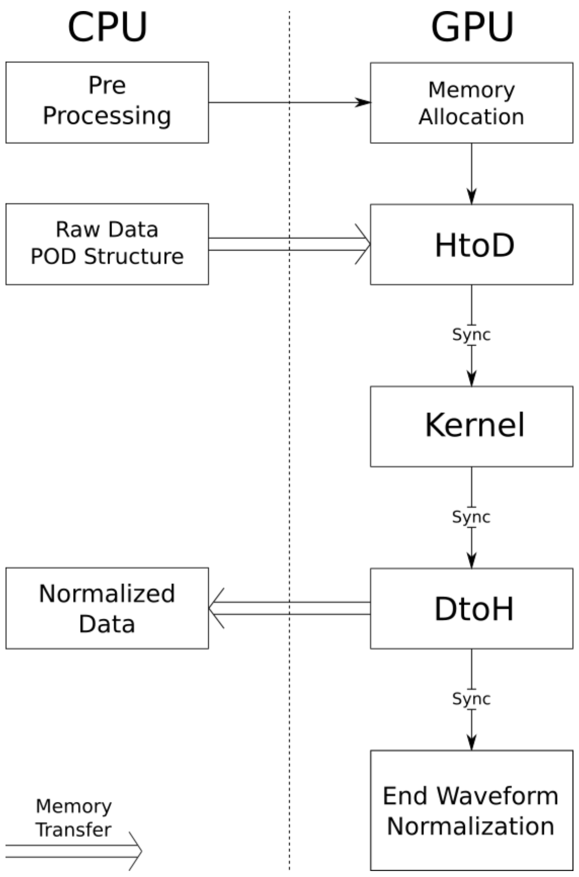


Figure 4.5: Workflow of the waveform normalization algorithm with synchronous memory transfers. Synchronization is forced between HtoD, Kernel and DtoH operations.

necessary number of streams is used (S_u). The number of streams used is calculated by dividing the total number of PODs (n_P) by the maximum number of threads per SM (m_T) (eq. 4.2). This latter value affects the number of PODs that can be processed concurrently. A further condition is applied to ensure that the number of used CUDA streams does not exceed the maximum number m_S.

$$S_u = min\left(\frac{n_P}{m_T}, m_S\right) \tag{4.2}$$

The block size (b_{size}) is determined based on device specifications, and chosen in order to maximize performance. Its value is specified in the beginning of the code and remains constant throughout its execution. The grid size (G_{size}) is determined based on the number of PODs, the number of used streams and the block size.

$$G_{size} = \frac{n_P}{S_u \times b_{size}} \tag{4.3}$$

The next step is allocating memory on the device to store the raw data and the POD information structure. For the host to device memory transfer (HtoD), kernel

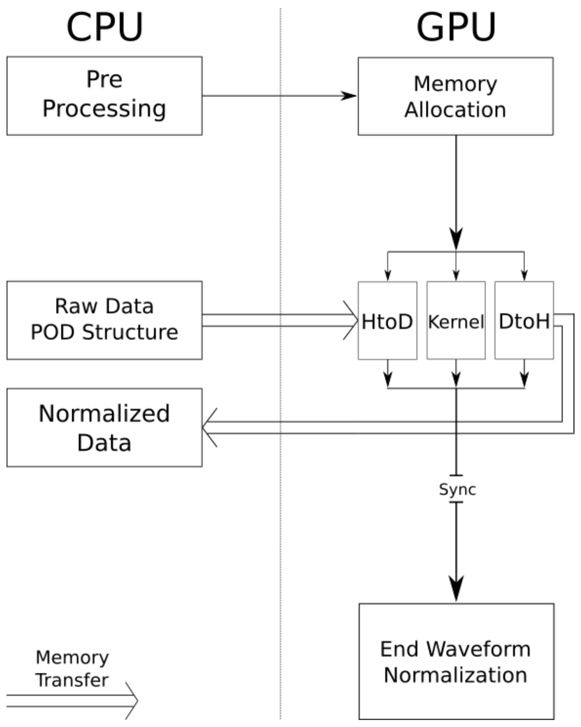


Figure 4.6: Workflow of the waveform normalization algorithm with asynchronous memory transfer. In this case the HtoD, Kernel and DtoH operations are launched simultaneously and an overlapping between them may occur. The synchronization is only forced at the end of their execution.

execution and device to host memory transfer (DtoH) two solutions (synchronous or asynchronous) were implemented, and the choice between them depends on the characteristics of the GPU board.

The synchronous solution performs each individual action one after the other, synchronizing all device threads between different actions (Fig. 4.7). It is worst performance-wise but it has the advantage of working independently of the GPU board used.

The asynchronous solution takes advantage of the ability of some GPUs to pin already allocated memory with *cudaHostRegister*. Using pinned memory allows to perform asynchronous actions between SMs. For example, SM n can make a HtoD transfer of a portion of the data, while SM $n+1$ is running a kernel that processes a different portion of the data. However, the individual actions remain synchronous inside each SM (Fig. 4.8).

The effective result of using the asynchronous method is an overlap between HtoD transfers, kernel executions and DtoH transfers happening in different SMs. A big performance advantage is found with this method, but only GPUs with compute capability greater than 1.1 can use it.

Waveform normalization finishes by transferring the normalized data back to

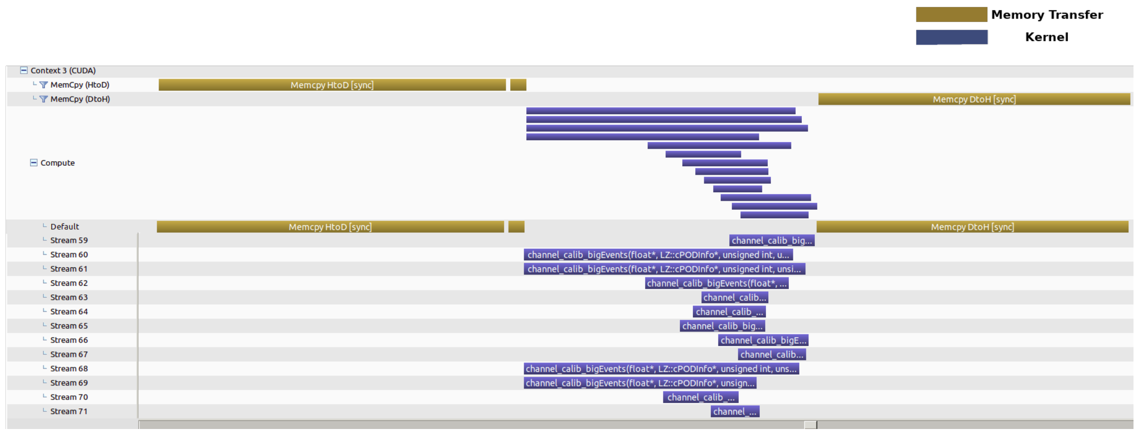


Figure 4.7: Image of the execution of the waveform normalization algorithm using pageable memory. In this case execution of HtoD, kernel and DtoH operations is sequential. The only overlaps that occur are between kernel executions in different CUDA streams. Image taken from the NVIDIA Visual Profiler.

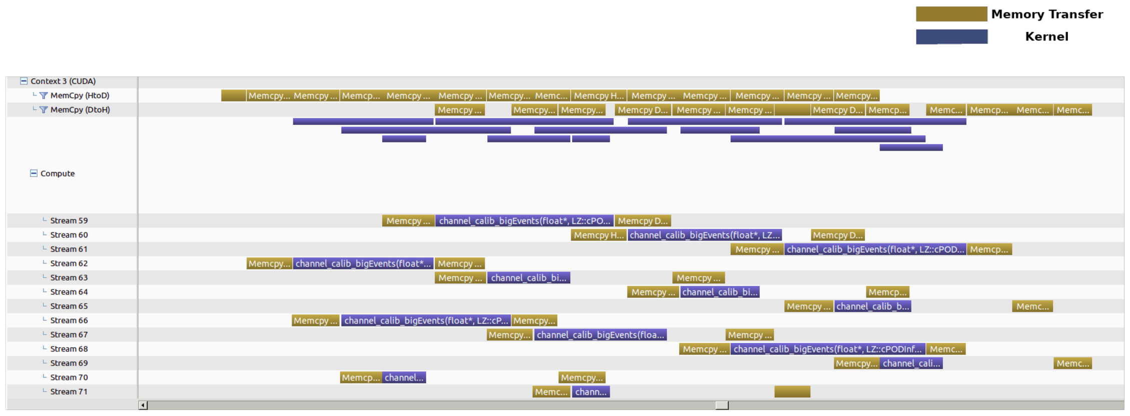


Figure 4.8: Image of the execution of the waveform normalization algorithm using pinned memory. Notice how the execution remains sequential inside a single CUDA stream, but between different streams overlaps may occur. Image taken from the NVIDIA Visual Profiler.

the host, but the memory remains allocated on the GPU for the waveform sum algorithm. Fig. 4.5 and 4.6 show the workflow of the version with synchronous and asynchronous memory transfers, repectively.

4.2.3 Kernel

The waveform normalization kernel starts by calculating the index of the POD being processed

$$i_{POD} = o + b_{idx} * b_{dim} + t_{idx}; \tag{4.4}$$

where o is the number of PODs processed by the previous threads, b_{idx} the index of the block in the CUDA grid, b_{dim} the number of threads inside a block and t_{idx} the index of the thread in the block.

After this, the information containing the POD offset, its size and the corresponding g and z is read from global memory. Then eq. 4.1 is applied to the data being processed by the thread. The normalized values replace the raw ones in the original array.

4.3 Waveform Sum

4.3.1 Overview

This algorithm takes the data produced by the waveform normalization, and sums it in order to reconstruct the total waveform of the event. An individual sum is created for each PMT array. The sums of the PMT arrays are then added together to give the total sum of each detector (Table 4.1). Figs. 4.9, 4.10 and 4.11 show examples of summed waveforms for the various detectors.

After separating the PODs into their corresponding detectors and PMT arrays, data is summed according to their absolute index in the event timeline. Although this absolute index is not stored for every element of the original array (to save space), it can be easily calculated from the POD idxStart, since the elements inside a POD are contiguous. Areas between pulses that have no data are padded with zeros, in order to have an accurate representation of the event timeline. The sums are also padded with zeros until they have a certain size defined in the configuration file. This final padding is related to the FFTs that are going to be used later on the pulse finding stage.

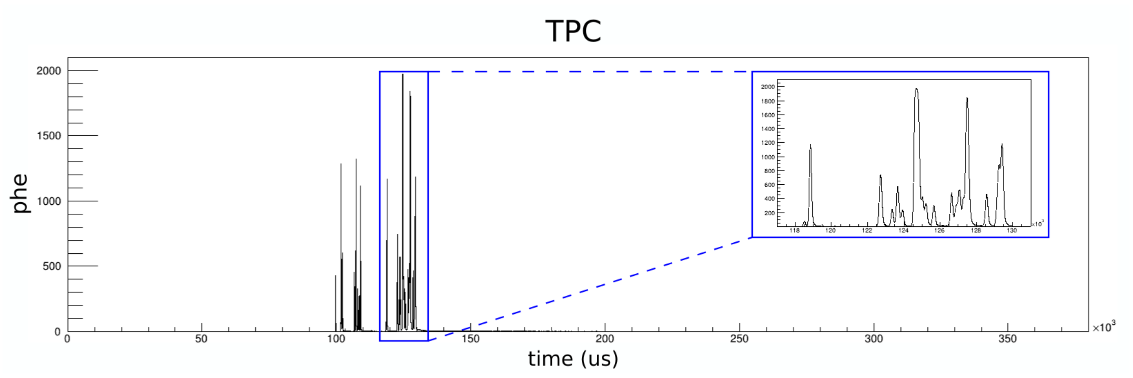


Figure 4.9: Example of a summed waveform for the TPC detector.

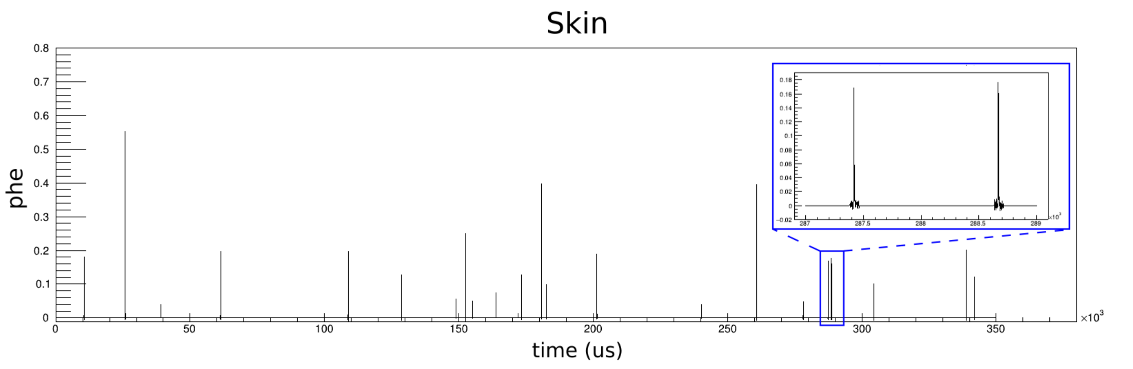


Figure 4.10: Example of a summed waveform for the Skin detector.

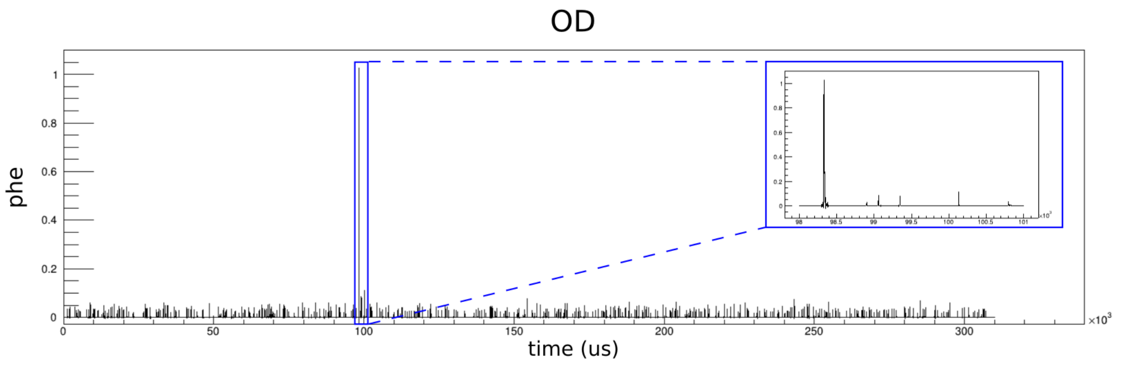


Figure 4.11: Example of a summed waveform for the OD. Here it is possible to see why the OD is such an important addition to LZ, because of the large amount of small pulses that "fill" the waveform. By detecting the interactions leading to these small pulses in the OD, it means that the particles that caused them will not reach the TPC or will be tagged if they do reach the TPC, drastically reducing the background of the experiment.

Similar to the previous algorithm, two solutions were considered. The first solution (WS1) was based on the absolute position of the data in the final summed waveform. These positions would be calculated during the normalization, based on the POD idxStart, and stored in an array. The calculated positions would then be used to sort the data on the normalized array, so that the elements corresponding to the same position are stored contiguously[3]. Then the final waveform would be divided into various segments, and each segment would be assigned to a different GPU thread. Each thread would sum the elements corresponding to its segment. The advantage of WS1 is that all threads can sum their elements independently, because there are no race conditions when writing to the final summed waveform. However, there are several drawbacks: the additional time (and memory) spent on creating the absolute positions array; the time spent on sorting the data; it is a complex code and its logistics are not simple.

The second solution (WS2) was based on the PODs. Each thread would be assigned to a different POD, and the values of its elements would be summed to their respective final summed waveform. The main issue with WS2 is the possibility of race conditions. The CUDA programming model does not allow concurrent writing by multiple threads to the same memory address. It ensures that at least one thread can write its value, but does not guarantee that all threads can write their values. So, the memory access needs to be serialized across the conflicting threads, to guarantee that all threads are able to write their value into the memory address. This serialization is achieved by using atomic operations.

This type of operations allow a thread to write its value into a specified memory location, while also ensuring that no other threads can access the same memory address at the same time. If a different thread tries to write its value into the same location, it will be held in a queue and wait until the previous thread has finished its writing. Cases where different locations are accessed at the same time are not altered by this function, and their writes remain concurrent.

The main advantage of WS2 is that it does not need the data to be sorted, which ”erases” the time spent on creating the array with the absolute positions and sorting it from the overall running time of the algorithm.

Ultimately, WS2 was chosen. In the tests performed, it was observed that sorting was a major limitation of the algorithm. The time it took to sort both the absolute index and normalized data arrays was, in the majority of the cases, larger than the time it took to perform the sum itself. So a ”sorting free” solution performs better

[3]note that as the normalized data would already be transferred back to the host prior to these transformations, they would not affect the representation of the data further ahead

by a factor of 5 to 10× (see Fig. 5.2 in Section 5.3).

As for the memory transfers they were kept synchronous. The fact that PODs are ordered on the normalized data array by channel and not by their starting index, means that there is no way to know when a portion of the data is ready to be copied back to the host, so asynchronicity of memory transfers is not a viable option.

4.3.2 Workflow

This algorithm is a direct follow up to the waveform normalization. The first step is iterating over the POD information structure and register which PMT arrays have data as well as the index limits of each of them. These index limits are then used to divide the normalized data array based on their respective PMT arrays. Fortunately, all the PODs that belong to a particular PMT array are stored contiguously in the normalized data array, so there is no need to rearrange it.

To store the sums, two arrays were allocated. One of those arrays is used to store the sum of one of the PMT arrays (array A), while the other stores the sum of the respective detector (array D). As said before, the size of these arrays is fixed and defined in the configuration file.

The kernel was specifically designed to only ever need a maximum of two arrays at the same time in order to perform the sum operations. This allows for a better optimization of memory usage. Sums are performed in the following manner: firstly the TPC (HG and LG), then the Skin and finally the OD (HG and LG).

The TPC and Skin sums are treated differently from the OD ones. Since TPC and Skin both have multiple PMT arrays, a version of the kernel that takes arrays A and D simultaneously is used (left branch of Fig. 4.12). The kernel then sums the normalized values into their respective positions in arrays A and D. When the sum of a PMT array is completed, array A is copied to host memory, its values are reset to zero and it is then reused to sum the next PMT array. Array D is only copied to host memory when the sums of all its PMT arrays have been completed. After being copied to host, array D is also reset and reused for the next detector.

The OD sums use a version of the kernel that takes only array D (right branch of Fig. 4.12). The process is then similar to the one described before.

4.3.3 Kernel

Like for the waveform normalization each thread starts by calculating which POD will be processed, using eq. 4.4.

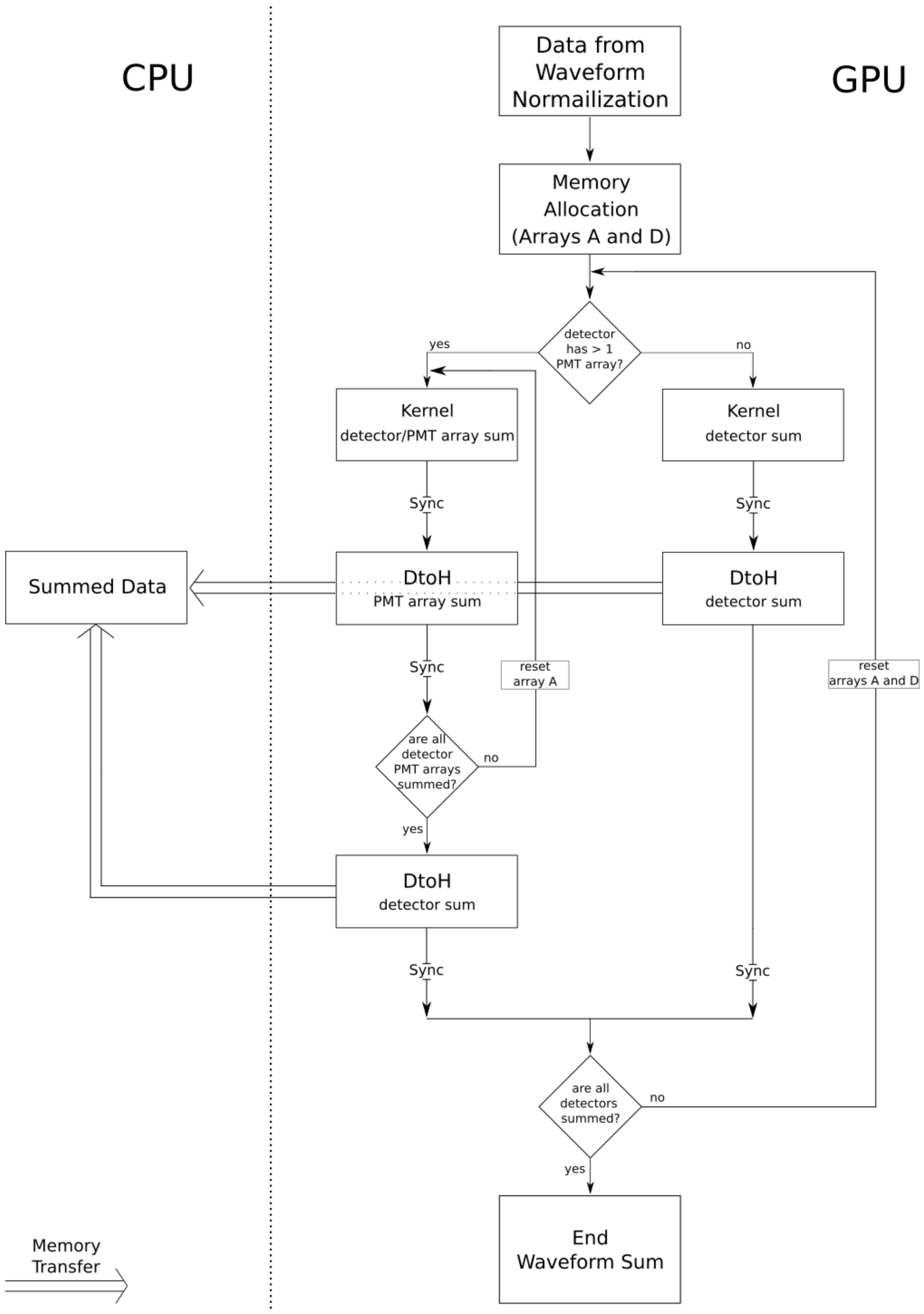


Figure 4.12: Workflow of the waveform sum algorithm.

It then reads, from the POD information structure (mentioned in Section 4.2.2), the values corresponding to the POD offset, POD idxStart and POD size. Then the normalized values of a POD are summed based on their absolute index, calculated based on the POD idxStart.

The atomic function used to serialize concurrent memory writes is *atomicAdd*. This function sums a value into a specified memory address.

The kernel has two different versions. The first version is designed to deal with detectors that have more than one PMT array (in this case TPC and Skin). This version adds the value to the PMT array sum, while simultaneously adding the same value to the detector sum. The second version deals with detectors that have only one PMT array (OD), only adding the value to the detector sum.

4.4 Pulse Parameterization

4.4.1 Overview

Pulse parameterization receives as input the pulses found by the pulse finder. The quantities that are calculated include: areas, fixed fractions and running integrals. Besides these, an extra quantity is also calculated for TPC pulses called Top-Bottom Asymmetry (TBA).

The areas are calculated by iterating over the elements of the pulse and summing those elements together. Four different areas are calculated: total area, which sums all the elements of the pulse; positive area, which sums the elements of the pulse whose value is positive; negative area, which sums the elements of the pulse whose value is negative; and squared area, which sums the squared values of all the elements in the pulse. The total, positive and negative areas can be later used to determine the origin of the pulse (S1, S2, afterpulsing, etc.). The squared area can be used to calculate the Root Mean Square (RMS).

The running integrals are also calculated during this iteration stage. There are two running integrals: one for the total pulse area, and the other for the positive pulse area. These integrals contain the total and positive areas of a pulse for each moment of the pulse, making them extremely useful for calculating the fixed fractions.

Two different fixed fractions are computed: fixed area fractions and fixed time fractions. Fixed area fractions give the time instant, within a pulse, where its positive area reaches a certain percentile (1%, 5%, 10%, 90%, 95% and 99%). Since it is based on the positive area, it uses the positive area running integral to calculate those fractions. The reason for this lies with the fact that this integral is guaranteed to be monotonically increasing, since it only accounts for the positive values of the pulse. This means that there is a single, univocal, time instant where the area is equal to one of the desired area fractions.

The fixed time fractions give the value of the total area of the pulse for specified time instants (50, 100, 200 and 500 ns). The total area running integral is used in this case and the values of each fraction are calculated by reading the integral value for each of the time instants.

Finally, the TBA is a TPC specific quantity that relates the areas of corresponding pulses in the top and bottom TPC PMT arrays. It is given by the ratio of the difference between the areas a_T and a_B of the top and bottom pulses, over the total area A of the pulse (eq. 4.5). This value is useful to determine if a pulse is a S1 or a S2.

$$TBA = \frac{a_T - a_B}{A} \tag{4.5}$$

When developing an implementation of this algorithm for the GPU an important factor had to be taken into consideration: the length of the pulses[4]. It was observed that, across all detectors, some pulses were much larger than the average length of the pulses. As some of the quantities to be calculated require iterating over all elements of the pulse (namely areas, integrals and the TBA), the presence of these larger pulses can have a detrimental effect on performance, if they are not correctly dealt with.

So, to achieve the maximum performance, parameterization is divided in two steps: calculation of the areas, integrals and TBA and calculation of the fixed fractions. Each of the steps is approached differently.

For the calculation of the fixed fractions, one CUDA thread is assigned for the parameterization of one pulse. Because these quantities are not calculated by iterating over the entire pulse, the number of memory accesses needed is relatively low and, more importantly, is independent of the pulse length. This means that each thread will have roughly the same amount of computational work to perform, meaning that for the majority of the execution time of the algorithm the GPU is being used close to its full potential.

For the calculation of the areas, running integrals and the TBA a different approach is required. Since the calculation of these quantities requires iterating over the pulses, the processing time of each pulse is determined by its length. Because of this, the previous solution cannot be applied efficiently to this step. The existence

[4]length here relates to the duration of the pulse

of a possible set of much larger pulses meant that the threads dealing with them would finish their execution much later than the ones processing the majority of the smaller pulses. This greatly hinders performance because most of the threads would be left idle while waiting for the larger pulses to finish processing, essentially wasting the parallel computing potential of the GPU.

So, to overcome this situation, the calculation of the areas, running integrals and the TBA is performed by assigning one thread block to one pulse. Each launched thread calculates the mentioned quantities for a portion of the pulse. This allows to distribute the work more evenly across all threads, meaning that no thread is kept idle during long periods of time.

4.4.2 Workflow

Parameterization starts by creating a structure with information about the pulses. That information comes from the pulse finder and consists on the length of the pulse and its starting index in the data array.

Then the algorithm copies the detectors waveform data into GPU global memory and allocates two global memory arrays to store the running integrals, with sizes equal to the length of the pulse data copied into the GPU. Then the parameterization kernels are called. First, the kernel that calculates the areas, the running integrals, and in the case of the TPC also the TBA, is executed. Then the second kernel calculates the fixed fractions. The grid size used is different for each kernel. For the first kernel, the grid size is equal to the number of pulses being parameterized. As for the second kernel, the grid size is determined by dividing the number of pulses by the block size, which is determined based on the length of the pulses. If the data contains a pulse whose length is very large, a block size of 128 threads is used, in all other cases a block size of 32 threads is used. Finally, all the calculated quantities are copied back to the host.

The full workflow of the algorithm is represented in Fig. 4.13.

4.4.3 Kernel

As was discussed in Section 4.4.1 the parameterization of the pulses is performed in two steps, with each step corresponding to a different kernel.

Pulse areas, running integrals, and the TBA (for the TPC pulses) are calculated by the first kernel. As was stated before, in this case each pulse is assigned to a different thread block. The elements of the pulse are divided equally across all

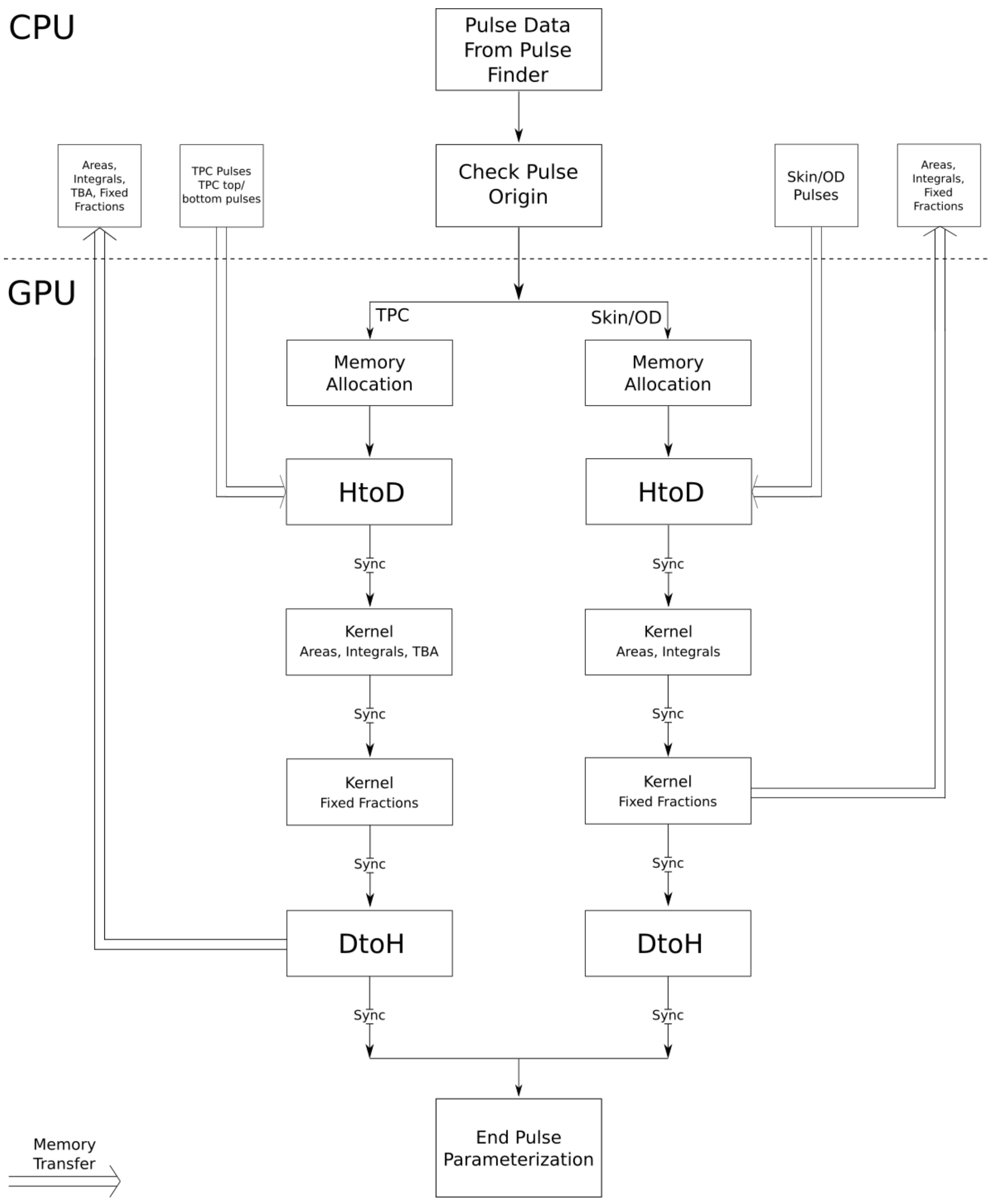

CPU
Pulse Data From Pulse Finder
Areas, Integrals, TBA, Fixed Fractions
TPC Pulses TPC top/ bottom pulses
Check Pulse Origin
Skin/OD Pulses
Areas, Integrals, Fixed Fractions
GPU
TPC
Skin/OD
Memory Allocation
Memory Allocation
HtoD
HtoD
Sync
Kernel Areas, Integrals, TBA
Kernel Areas, Integrals
Kernel Fixed Fractions
Kernel Fixed Fractions
DtoH
DtoH
End Pulse Parameterization
Memory Transfer

Figure 4.13: Workflow of the parameterization algorithm.

threads in the block.

Each thread will iterate over its portion of the pulse and calculate its total, positive, negative and squared areas. For the TPC pulses the top and bottom areas are also calculated. Then those values are written into shared memory arrays, so that all threads can access them at a later stage. There is a total of 4 (for Skin/OD pulses) or 6 (for TPC pulses) shared memory arrays, one for each of the areas mentioned, and all have the same size, which is equal to the number of threads inside the block.

During the iteration stage each thread also creates a total and positive running integrals for each portion of the pulse. To obtain the complete running integrals of the pulse each thread needs to correct the integral of its portion based on the values of total and positive areas calculated by the previous[5] threads. Since those values are already available on the shared memory arrays this is easy to achieve.

The final values for the total, positive, negative, squared, top and bottom areas are obtained by performing a parallel reduction[6] over each shared memory array.

The TBA is calculated using eq. 4.5 and the squared area A_S is calculated using eq. 4.6, where P is the length of the pulse, A the total area and e_i the element at index i of the pulse.

$$A_S = \sqrt{\frac{\sum_{i=0}^{P} e_i^2}{P} - \left(\frac{A}{P}\right)^2} \tag{4.6}$$

Finally, the first thread of the block writes all the calculated quantities into the global memory.

The second kernel calculates the fixed fractions. In this case, a thread is used to compute the fixed fractions of a pulse. For the fixed area fractions, the thread scans the positive area running integral looking for the instants where the percentage thresholds presented in Section 4.4.1 occur. When it finds them they are written into the global memory. If a percentage threshold falls between two consecutive time instants a linear interpolation is performed to approximate the real value for the time instant.

For the scanning of the integral, a binary search is used. This is a very effective method since it only requires an average of $\log_2 n$ accesses to global memory, with n being the length of the integral. To further increase performance, the percentage thresholds are searched in descending order (starting from 99% and ending on 1%)

[5]previous here relates to the index of the thread and not its instant of execution. For example, thread 0 is considered to be previous to thread 1 even though they execute simultaneously.

[6]a parallel reduction is a type of parallel algorithm commonly used to sum all the elements of an array. The elements are iteratively summed in pairs until the total sum is stored on the first element of the array.

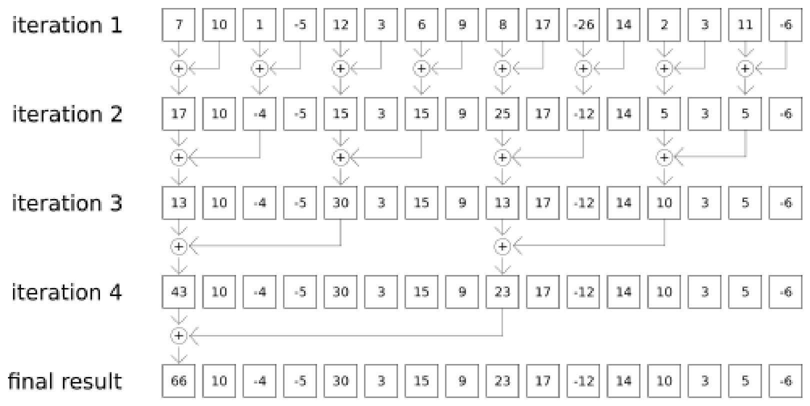


Figure 4.14: Example of a parallel reduction algorithm used to sum the elements of an array. In each iteration the sum operations are performed simultaneously, which means that they can be considered a "one sum operation" computation-wise. The result is that instead of having to perform n operations to sum the n elements (like in a sequential form), the parallel reduction only needs $log_2 n$ operations to sum all the elements.

so that the upper limit of the binary search can be reduced based on the results of the previous search.

The fixed time fractions are calculated by simply accessing the element of the total area running integral corresponding to the desired time instant. The only verification needed is to check if the time instant does not exceed the length of the pulse. In the cases that it does the last element of the integral is used instead.

4.5 Position Reconstruction

4.5.1 Overview

The final algorithm of the GPU-based signal processing chain is the position reconstruction. Unlike all previously described algorithms, this one only runs for pulses coming from the TPC Top PMT array. Its purpose is to estimate the x and y coordinates of an event, based on the hit pattern produced by the S2 pulse in the TPC Top PMT array.

The CPU implementation of position reconstruction used in this work is based on the standalone version of the LZAP Mercury module [67] (code available at [68]) and the GPU code was ported from ANTS2, an open source simulation and data processing package for scintillation detectors [69] (code available at [70]).

The reconstruction of the events is performed by scanning the XY parameter space in order to find the best match between the signals acquired by the PMT array, and the signals expected from the PMTs Light Response Functions (LRFs).

LRFs are functions that describe the probability of a given PMT to detect a photon originating at a given position inside the detector. Since the light collection of the PMTs depends on their position in the PMT array, each PMT will have its individual LRF. The process used for LRF determination is described in [71]. They are modeled as axial functions originating from the corresponding PMT axis. Their parameterization is based on cubic splines, whose coefficients are stored in the form of an array.

The reconstruction algorithm is implemented on the GPU by assigning a block of threads to one event. Each block uses a method called "contracting grids" to reconstruct the position of its respective event.

For each event a grid of nodes is created. The distance between them is determined by a scale factor defined in the configuration file. For each of the nodes, the difference between the fixed measured signal and the one expected by the LRFs is calculated, assuming that the S2 signal originates from the node being evaluated. Then, the node which minimizes this difference is found and a new grid is created centered on said node.

The distance between the nodes of this new grid is determined by dividing the distance between the nodes on the previous grid by a scale reduction factor also defined in the configuration file. This means that, with every iteration, the grid of nodes contracts around the position where the S2 signal originated.

The described process continues until a maximum number of iterations is reached. The node that gives the minimum difference of the last grid is considered to be the position of the S2 signal. The algorithm also produces an estimation of the energy of the interaction that originated the pulse.

One crucial point of the algorithm is determining the initial position for the center of the grid. There are multiple ways to determine that position. The method chosen for this project was calculating the center of gravity (CoG) of the signals of all PMTs, as it proved to be the most effective approach. Other possible methods are choosing the initial position to be the center of the TPC Top PMT array, or centering the initial grid on the PMT with the strongest signal.

4.5.2 Workflow

Since not all S2 pulses need to be reconstructed, the position reconstruction algorithm starts by selecting which pulses are to be reconstructed. This selection is based on the area of the pulse, calculated previously by the pulse parameterizer. The pulses with the larger area are the ones selected, because the smaller pulses usually come from S1 signals and noise sources, like S2-tails[7].

After this, the selected pulses are checked for saturation of the ADCs. By default, the signal from the HG channel of the TPC Top PMT array is used. In the cases where the channels saturate, the LG channel needs to be used, otherwise the performance of the reconstruction would be compromised.

Firstly, it is checked if any of the TPC Top PMT array channels saturates during the selected pulse. If this does not occur, no further checks are necessary and the signal from the HG channel can be used. If there are saturated PMTs the algorithm proceeds to check which, and substitutes the HG signal by the LG signal of those channels. This allows to obtain an optimal signal-to-noise ratio for every pulse being reconstructed.

After checking for saturation, the signal acquired by each PMT during the pulse duration is calculated and stored in an array. Before sending this array to the GPU, the PMT signals are used to calculate the starting position of the reconstruction.

As referred before, this starting position is defined by the CoG of the pulse. It is calculated by summing the product of the PMT signal by its x and y coordinates, and then dividing that sum by the sum of the signals from all PMTs. Only the PMTs whose signal is above a defined cutoff threshold (10% of the strongest PMT signal) are used for this calculation. Eqs. 4.7 and 4.8 show the formula used to calculate the CoG of the pulse, with s_i, x_i and y_i being the signal, x and y coordinates of PMT i. S is the cutoff threshold.

$$x_{CoG} = \frac{\sum_{i=0}^{n} s_i x_i}{\sum_{i=0}^{n} s_i}, \qquad s_i > S \tag{4.7}$$

$$y_{CoG} = \frac{\sum_{i=0}^{n} s_i y_i}{\sum_{i=0}^{n} s_i}, \qquad s_i > S \tag{4.8}$$

The resulting CoG coordinates are added to the array with the PMT signals, and the array is sent to the GPU. The LRFs and the PMT coordinates are read from a JSON file at the start of the algorithm. All this data is stored in GPU global memory and the reconstruction kernel is launched. The size of the CUDA grid is equal to the number of pulses being reconstructed. Unlike in previous algorithms, the thread blocks launched here are two-dimensional, to better account for the 2D complexity of the problem. The number of threads in each block is 64, which means that there are 8 threads in the x dimension of the block and 8 threads in the y dimension. Since each thread corresponds to a node of the reconstruction grid, this means that the grid has 64 nodes, 8 in each dimension.

[7]late extraction of electrons trapped by impurities or under the surface of the liquid

After the kernel execution is complete the coordinates of the reconstructed pulses are sent back to the host. An estimation of the pulse energy and the χ^2 value associated with the result are also sent back to the host, to give further insight into the quality of the reconstruction.

The workflow of the algorithm is presented in Fig. 4.15.

4.5.3 Kernel

The position reconstruction kernel takes full advantage of the GPU shared memory to improve the performance of the reconstruction. Shared memory arrays are created to store the coordinates of the PMTs, the signals of the PMTs, the IDs of the nodes, the values of χ^2 of each node and the estimation of the pulse energy.

The first step consists in loading the coordinates of the PMTs and their signals from global to shared memory. This is advantageous because all threads inside the same block will use the same data and also because each thread will have to access this data multiple times in each iteration.

Each node then computes the total signal measured by the PMTs and the expected signal given by the LRFs, assuming S2 coordinates corresponding to the current node. These values are calculated by summing the squared differences between the measured signals and the expected signals, for all the PMTs that are within a certain radius (in this case 400 mm) from the node being evaluated.

Next, the χ^2 value of each node is calculated using eq. 4.9, where n is the number of PMTs that meet the selection criteria, s_i is the signal of PMT i, LRF_i is the expected signal given by the LRF of PMT i and E is given by eq. 4.10.

$$\chi_n^2 = \frac{1}{E}\sum_{i=0}^{n}\frac{s_i^2}{LRF_i} - \sum_{i=0}^{n} s_i \tag{4.9}$$

$$E = \frac{\sum_{i=0}^{n} s_i}{\sum_{i=0}^{n} LRF_i} \tag{4.10}$$

The χ_n^2 values are then compared against each other in pairs, until the minimum value of χ_n^2 is found. The values are compared in a parallel reduction scheme similar to the one shown in Fig. 4.14. The coordinates of the corresponding node are then used as the center of the new grid, with the distance between the nodes reduced by a scaling factor. The described process is illustrated in Fig. 4.16

The mentioned steps are then repeated for each iteration from the calculation of the node coordinates onwards, until the maximum number of iterations is reached.

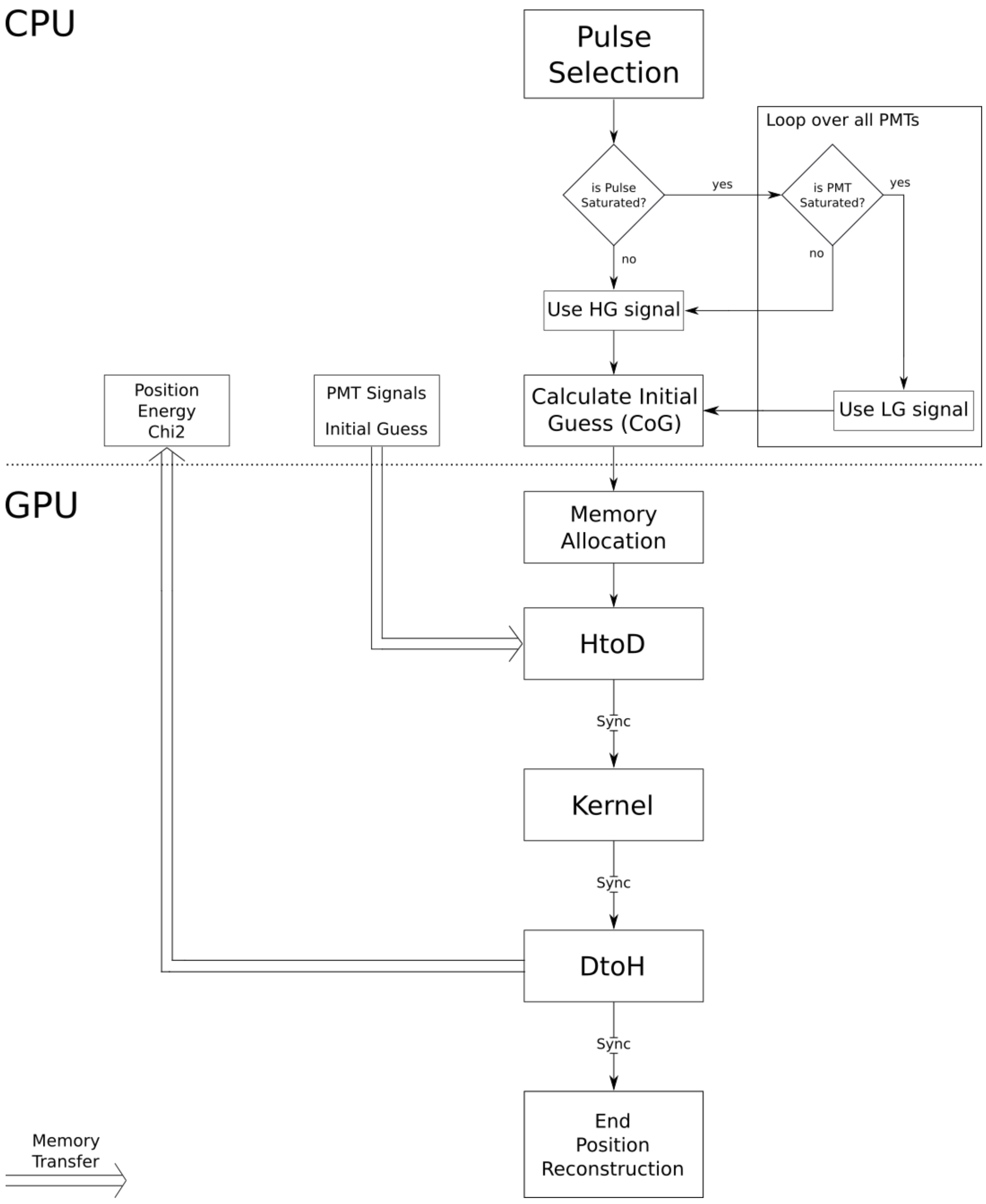


Figure 4.15: Workflow of the position reconstruction algorithm.

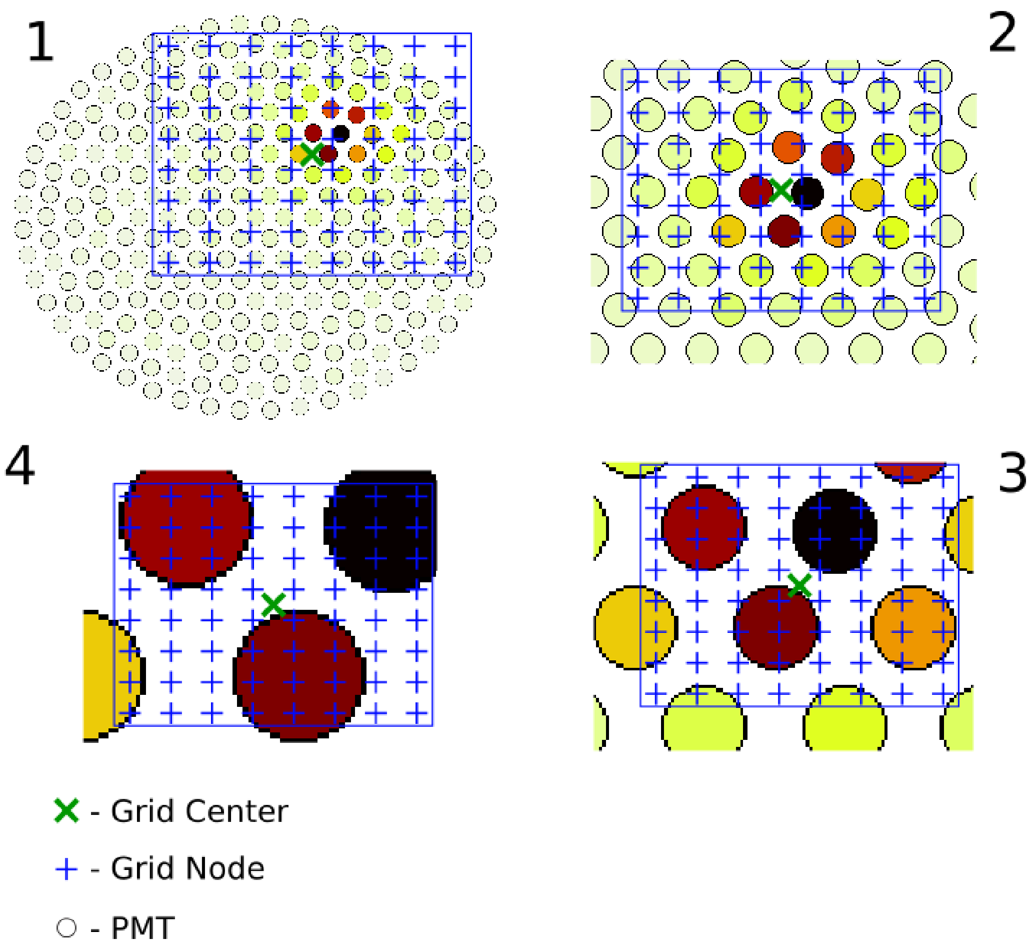


Figure 4.16: Illustration of the contracting grids algorithm with 4 iterations. The PMT signals range from white to black, with the darkest PMTs being the ones with the strongest signals. The grid center of image 1 is determined by calculating the CoG of the PMT signals. Images 2, 3 and 4 correspond to the iterations performed by the algorithm. Their grid centers are determined by the results of the previous iteration. After the last iteration is complete, the calculated next grid center corresponds to the reconstructed position.

When that happens, the values of the x and y coordinates of the node with the lowest χ^2 (reconstructed position), the χ^2 and the estimated energy of the S2 pulse are stored in GPU global memory.

5

Tests and Results

In this chapter, the tests performed and the results obtained are going to be presented and discussed. Two types of tests were conducted: numerical precision and performance tests.

Numerical precision tests are conducted with the objective of ensuring that the results of obtained by the GPU are within a certain tolerance of the ones obtained by the CPU. These tolerances result from the differences in floating point precision between CPU and GPU. For the waveform normalization and sum, and for the pulse parameterization, both results have to be exactly the same, since the implemented algorithms are similar. For the position reconstruction a larger tolerance is permitted because, as the CPU and GPU algorithms are not exactly the same, differences between their results are expected.

Performance tests have the objective of measuring the timing performance of the GPU algorithms. Their analysis, when compared to the CPU, allows to identify the performance gains obtained by the GPU and the types of events where these performance gains can be beneficial for the overall performance of the UPM.

5.1 Test Platform

All the development and testing of the algorithms was conducted on a computer at LIP-Coimbra with one Intel Core i7-6700 CPU, two NVIDIA GeForce GTX 970 GPUs and 16 GB of DDR4 DRAM working at 1066.5 MHz. The characteristics of the hardware are presented in Tables 5.1 and 5.2 respectively.

The tests were performed using only one CPU thread and one GPU, to simplify the analysis of the results. This analysis still remains valid when the number of CPU threads increases, because adding more CPU threads only introduces concurrency between the execution of different tasks and not parallelism inside a single task.

Intel Core i7-6700	
# of cores	4
# of threads	8
Clock frequency	3.4 GHz
Memory Bandwidth (max)	34.1 GB/s

Table 5.1: Characteristics of the CPU used [72].

NVIDIA GeForce GTX 970	
# of CUDA cores	1664 (128 per SM)
# of SMs	13
Global Memory	4 GB
L1 Cache/Shared Memory	48 kB (per SM)
L2 Cache	2 MB
Clock frequency	1050 MHz
Memory Bandwidth (max)	224.4 GB/s
CUDA compute capability	5.2

Table 5.2: Characteristics of the GPU used [73].

5.2 Simulation Data

The data used to test the developed GPU algorithms (and the UPM in general) comes from Monte Carlo simulations of the LZ events. These simulations serve multiple purposes: assessing the design features of the detector; calculating the rate of background events; predicting the sensitivity of the experiment; and simulating the complete electronic processing chain. The latter is particularly important for this project, as it allows to test the analysis chains and ensure that they are ready for the moment when LZ starts its data acquisition.

The chain that simulates the electronic processing in LZ has two main components: BACCARAT[1] and the Detector Electronics Response (DER) module. BACCARAT is a code developed within GEANT4 to simulate low-background experiments, such as LZ. The DER is a software package that simulates the PMT signal generation and the signal processing done by the LZ electronics [74].

The simulation process begins by generating particles and tracking them and their interactions across the LZ setup, using BACCARAT. The PMT hits recorded in this initial process are then passed to the DER. The DER simulates the response of the PMTs and the shaping and amplification of their signals. The simulated waveforms are outputted in an identical format to what is planned for the DAQ.

[1]Basically, A Component-Centric Analog Response to AnyThing

Files generated with different versions of the simulation tools are grouped in Mock Data Challenge (MDC) files [74].

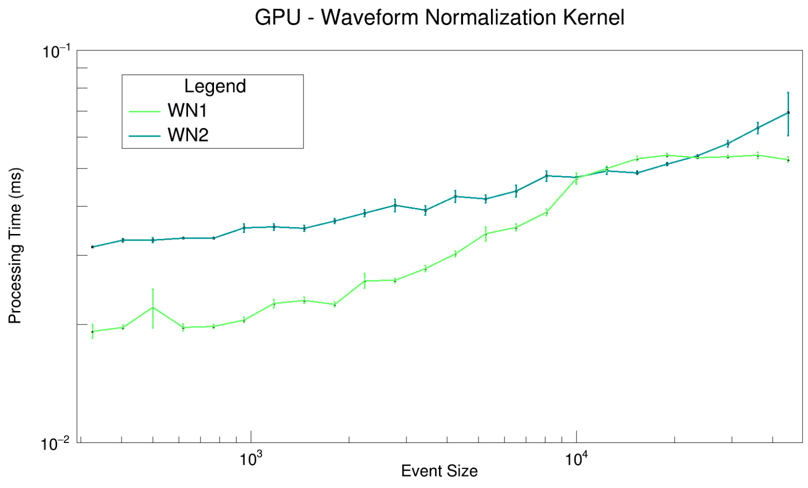
In this project, two sets of simulated data, from different interaction sources, were used for the tests. Both sets are from MDC3, the third iteration of the LZ simulations. One of the sets contains data from Deuterium-Deuterium (DD) neutron interactions and the other data from ^{83m}Kr. In LZ, monoenergetic DD neutrons with 2.45 MeV are used to calibrate the NR bands. They will reach the detector through one of two tubes that penetrate its side. The interactions of the ^{83m}Kr source are distributed uniformly through the detector [74]. Both these sources are good to calibrate the position reconstruction algorithm, because their event densities across the TPC are well defined.

5.3 Waveform Normalization and Sum

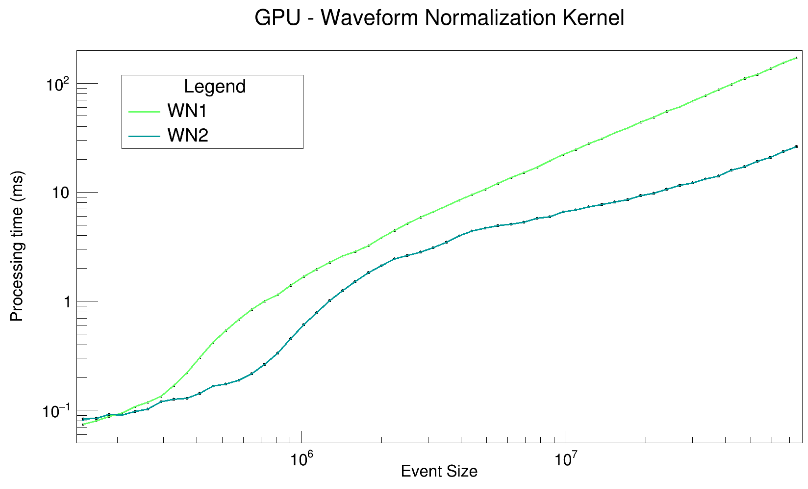
On the numerical precision tests, the GPU implementation obtains identical results to the CPU, within the standard floating point precision.

As was discussed in Sections 4.2.1 and 4.3.1, two implementations were considered for the kernels of both algorithms. The normalization algorithm implementation 1 (WN1) consisted in dividing the normalization work equally across all launched threads; implementation 2 (WN2) consisted in assigning 1 thread for the normalization of one POD (see Section 4.2.1 for more details about both implementations). In Fig. 5.1 the performance of both versions is compared. It can be observed that for smaller events ($<10^4$ elements on the raw data array) WN1 is slightly faster than WN2 (Fig. 5.1a). This can be simply explained by the fact that smaller events also have less PODs and, therefore, the number of active threads in WN2 is also less and far from the maximum potential of the GPU. Between 10^4 and 4×10^5 elements, both implementations are similar in performance.

For events with more than 4×10^5 elements a performance advantage in favor of WN2 becomes apparent. Until event sizes of 3×10^6 elements the performance difference remains roughly constant of about 3-4×, after that the difference starts to increase with the number of elements and the number of PODs, reaching performance differences of up to roughly 10× (Fig. 5.1b). The better performance of WN2 can be explained by a more efficient use of the threads, which allows to have the algorithm running closer to full GPU potential for a longer period of time while simultaneously reducing the number of global memory accesses performed by each thread.



(a) Small events

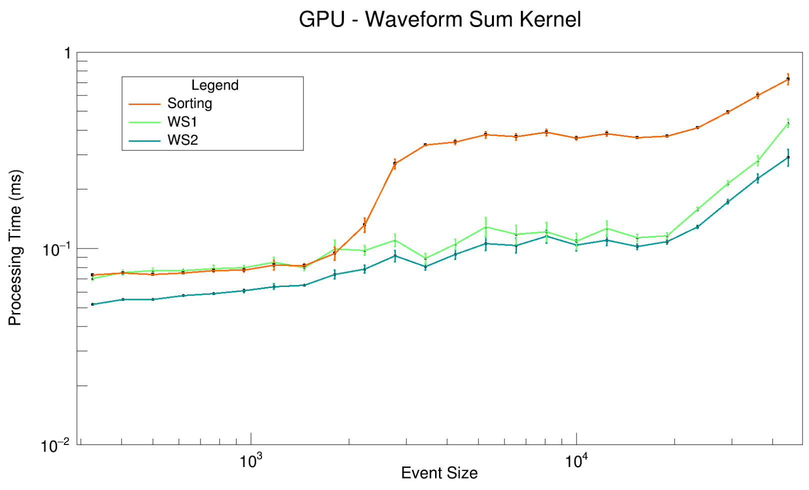


(b) Large events

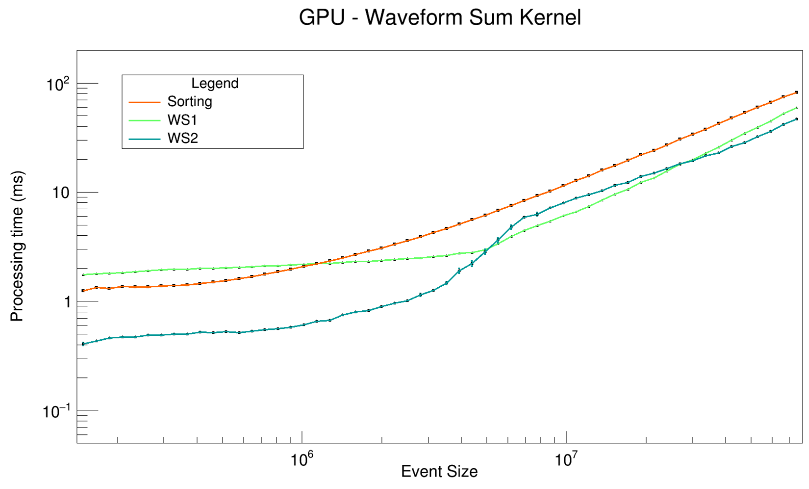
Figure 5.1: Plot of the processing time as a function of event size for both versions of the waveform normalization kernel.

Based on this analysis, WN2 was chosen for the waveform normalization kernel due to 2 main reasons: 1) its performance is significantly better than WN1 for most event sizes; and 2) in the only region where WN1 performs better, obtaining a performance gain by using the GPU will be virtually impossible (see discussion about memory transfer times further ahead).

Another aspect that was noticeable during the analysis was the existence of a gap without data between 5×10^4 and 1×10^5, which can be explained by the absence of data from the TPC detector up until that point. As TPC PODs are generally larger than the ones from other detectors, a jump in event size can be observed from the moment they start being present in the data. This gap was used to define what were considered small and large events: small events are the ones to the left of the



(a) Small events

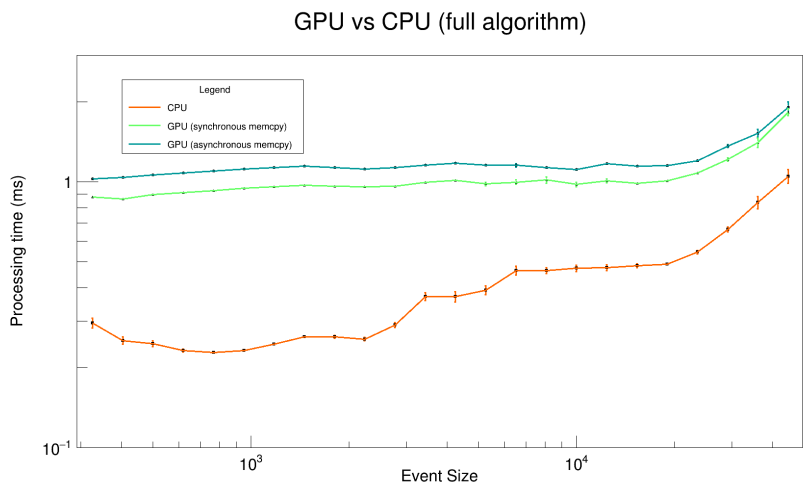


(b) Large events

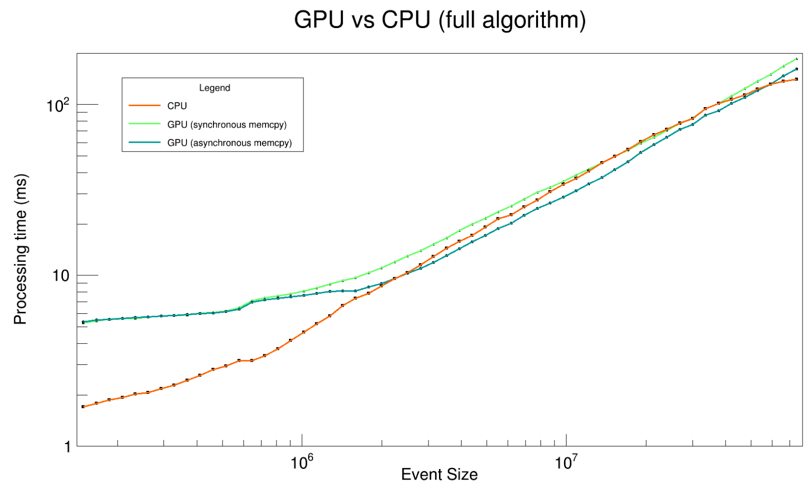
Figure 5.2: Plot of the processing time as a function of event size for both versions of the waveform sum kernel.

gap and large events are the ones to the right of the gap.

For the sum algorithm, implementation 1 (WS1) consisted on dividing the final waveform in segments and assigning one thread to sum each segment. This implementation never has race conditions when writing the sum results to memory, but requires the data to be sorted. Implementation 2 (WS2) follows a similar approach to WN2 by assigning a thread to each POD. In this implementation the data does not have to be sorted, but race conditions are very likely to occur when writing to memory. To solve this issue atomic operations are used to perform the sum of the elements (see Section 4.3.1 for more details about both implementations). Fig. 5.2 shows the processing time of both implementations as a function of the event size. The time spent on sorting the data is also shown to understand why that is a



(a) Small events



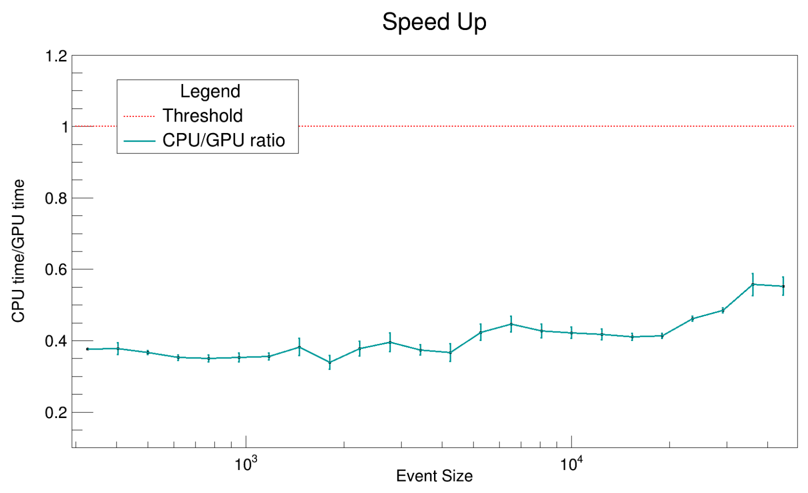
(b) Large events

Figure 5.3: Plot of the processing time as a function of event size for the full waveform normalization and sum algorithms. The processing times for the CPU and GPU, with synchronous and asynchronous memory transfers, are presented here.

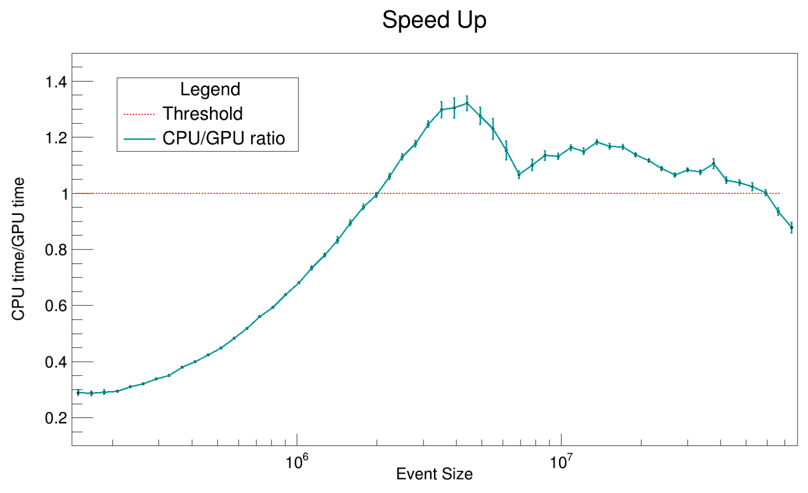
limiting factor for WS1.

With the exception of the range between 10^5 and $7{\times}10^6$, where there is a large difference, the performance of the algorithm is very similar for both implementations, but the performance of WS2 is generally better. The determining factor in choosing WS2 over WS1 is the fact that it doesn't require the data to be sorted. As can be seen from Fig. 5.2, the overall performance of the full algorithm would be severely limited by the time spent on sorting the data.

Regarding the full waveform normalization and sum algorithm, Fig. 5.3 compares the performance of the CPU implementation with the GPU implementations using synchronous and asynchronous memory transfers. The GPU processing times presented here account for memory allocation on the GPU, HtoD and DtoH trans-



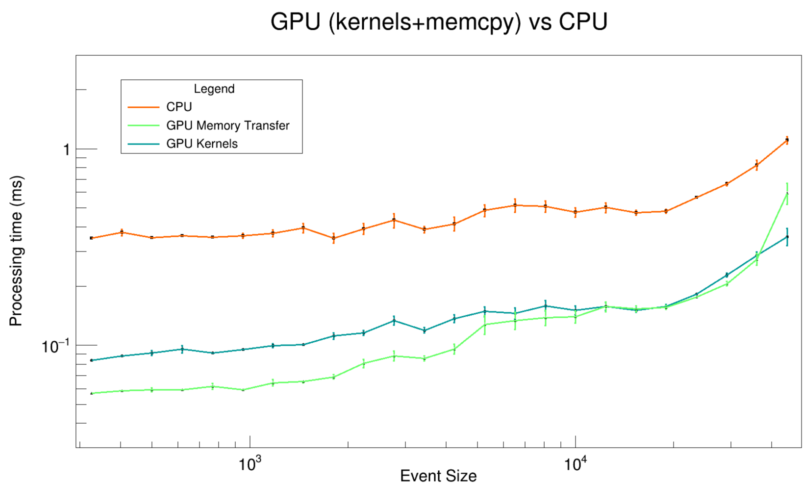
(a) Small events



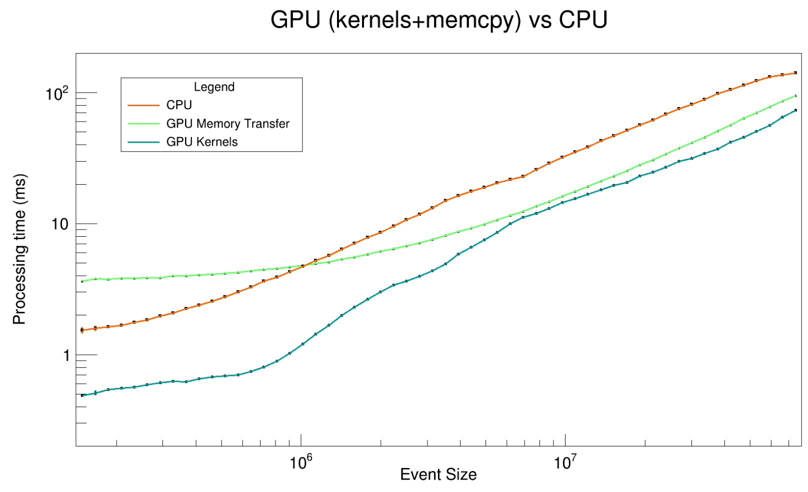
(b) Large events

Figure 5.4: Representation of the ratio between the CPU and GPU processing times for the waveform normalization and sum. The GPU implementation used to calculate this ratio is the one with asynchronous memory transfers. The dashed red line represents the threshold where the GPU is faster than the CPU, the events above it have a performance gain by running the waveform normalization and sum on the GPU.

fers and kernel execution times. When synchronous transfers are being used there are no performance gains by using the GPU. As for the asynchronous transfers, a small gain is obtained in the 10^6-10^7 event size range. The magnitude of this gain can be better observed in Fig. 5.4b. Unexpectedly, the performance of the GPU starts decreasing on the 10^7 range, being slower than the CPU for events larger than $7{\times}10^7$. This can possibly be explained by an increase on the time it takes to allocate and transfer these larger blocks of memory. Regarding the smaller events, the worse performance of the GPU relative to the CPU is to be expected due to the additional operations that need to be performed by the GPU (i.e. global memory allocations,



(a) Small events

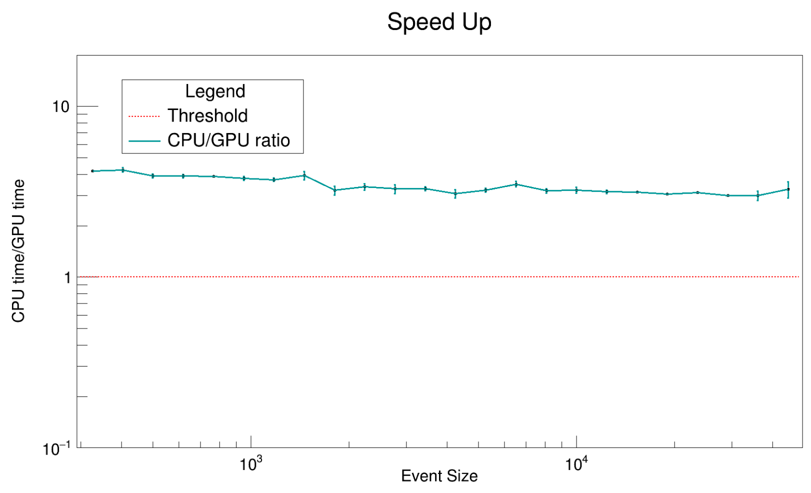


(b) Large events

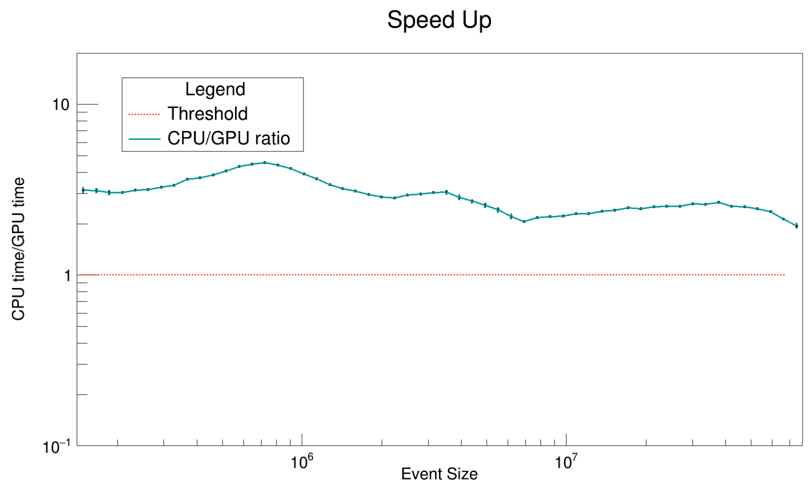
Figure 5.5: Plot showing the individual contributions of the kernel and the memory transfers to the processing time of the GPU algorithm for the waveform normalization and sum.

memory transfers, etc.).

To better understand why the GPU implementation fails to yield a performance advantage for some of the larger events the plot in Fig. 5.5 was produced. Here the individual contributions of the kernel and the memory transfers are compared against the full CPU algorithm. These contributions were obtained by timing separately the memory transfers and the kernels. It is possible to observe that the limiting factor is the time spent transferring the data to the GPU. This is more noticeable in the 10^5-10^6 range where this action alone takes more time than the entire CPU algorithm. Another conclusion that can be taken from this plot is that the performance of the GPU kernel is always significantly faster than the CPU algorithm, a fact that is more visible in Fig. 5.6. This gives an estimation of the



(a) Small events



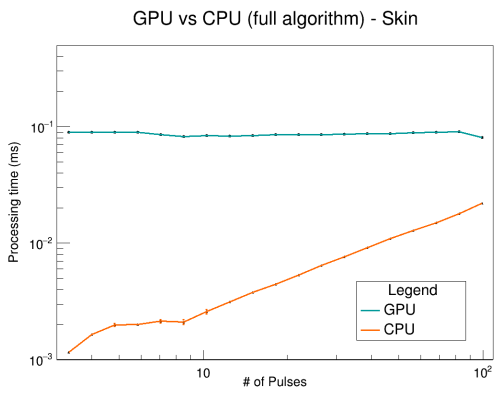
(b) Large events

Figure 5.6: Representation of the ratio between the CPU and GPU kernel processing times for the waveform normalization and sum. The dashed red line represents the threshold where the GPU kernel is faster than the CPU. As shown in the plots the performance gain given by the GPU kernel remains roughly constant for all event sizes.

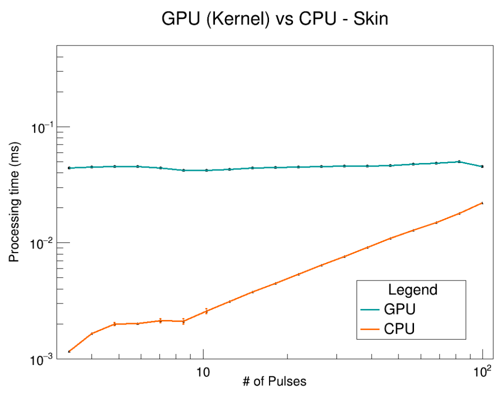
maximum potential that can be achieved by the GPU implementation, if a more efficient way of transferring the data to the GPU is found.

5.4 Pulse Parameterization

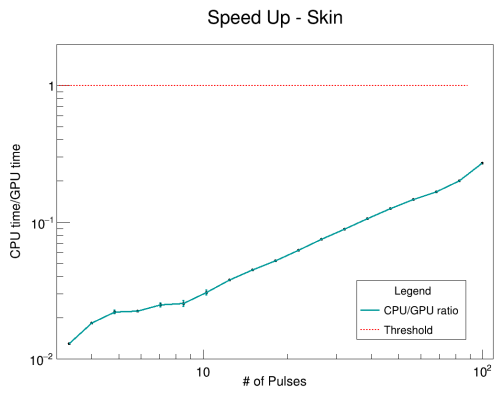
Regarding pulse parameterization, the numerical precision tests performed yield very similar results to the ones for the previous algorithm, with identical results to the CPU within the standard floating point precision.



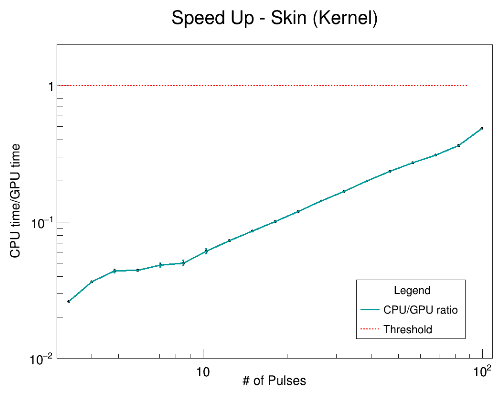
(a)



(b)



(c)

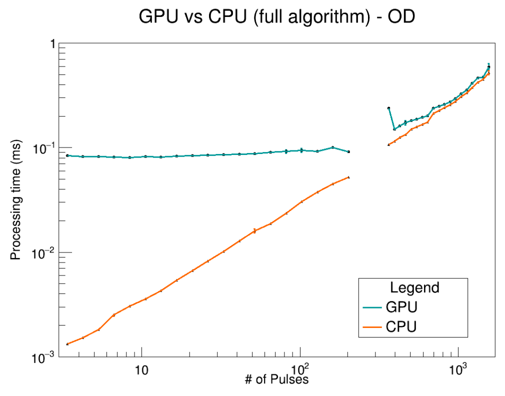


(d)

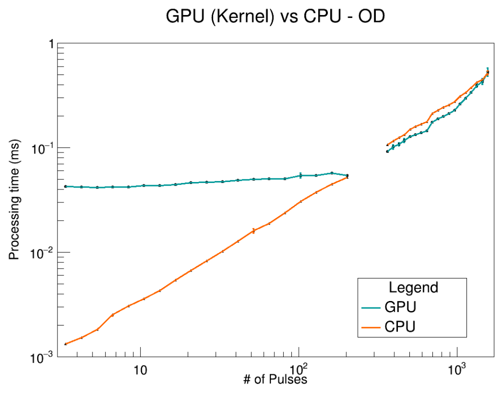
Figure 5.7: Plots showing the performance of the pulse parameterization algorithm for Skin pulses. Figures a) and b) show the processing time as a function of the number of parameterized pulses for the full GPU algorithm and for the GPU kernel respectively, when compared to the full CPU algorithm. Figures c) and d) show the ratio between the CPU and GPU curves in plots a) and b). The red dashed line in these plots represents the threshold above which the GPU algorithm is advantageous performance-wise.

As for the performance tests, the processing time of the algorithm in various scenarios was measured, in order to determine the key parameters driving the performance gain. The analysis of the results was performed independently for each detector, due to the fact that the average pulse length is different between them and the average number of pulses per waveform also differs.

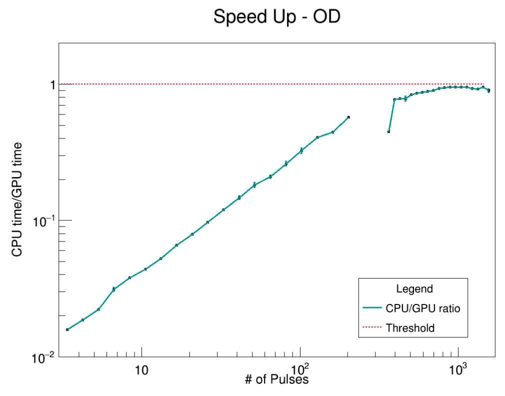
The easier results to analyze are the ones from the Skin detector. As can be seen in Fig. 5.7 there is no real advantage in parameterizing them in the GPU, in fact the CPU performs approximately 10× better than the GPU. This can be explained by a number of reasons. The most important one is related to the length and number of pulses in the Skin. The number of pulses ranges from 2 to 100 and the average length ranges from 15 to 20 elements (150-200 ns). As both these quantities are low, the full processing potential is not reached, something that is clearly visible in



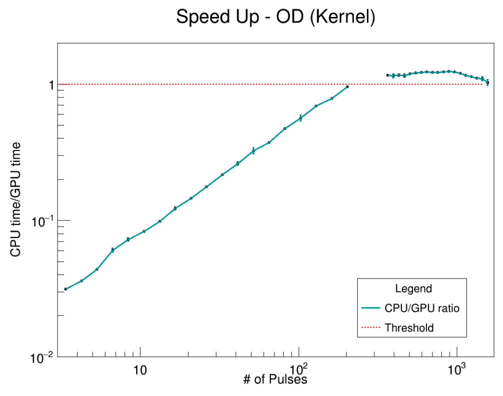
(a)



(b)



(c)

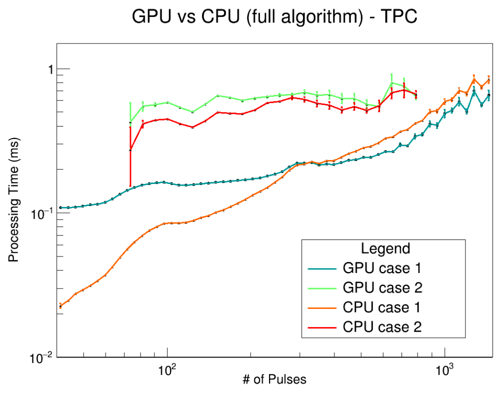


(d)

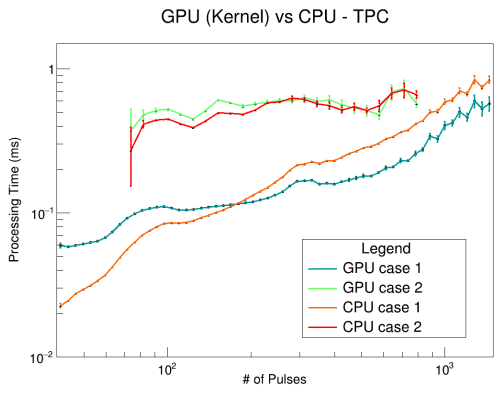
Figure 5.8: Plots showing the performance of the pulse parameterization algorithm for OD pulses. Figures a) and b) show the processing time as a function of the number of parameterized pulses for the full GPU algorithm and for the GPU kernel respectively, when compared to the full CPU algorithm. Figures c) and d) show the ratio between the CPU and GPU curves in plots a) and b). The red dashed line in these plots represents the threshold above which the GPU algorithm is advantageous performance-wise.

Figs. 5.7a and 5.7b by the fact that the processing time remains roughly constant with the increase of the number of pulses. Another important factor is related to the GPU kernel launch time which is, on average, 0.02 ms. By taking a closer look at the y-axis of Figs. 5.7a and 5.7b, and considering the fact that two kernels are launched per parameterized waveform, it is possible to see that the time spent on launching the kernel alone is longer than the full CPU algorithm.

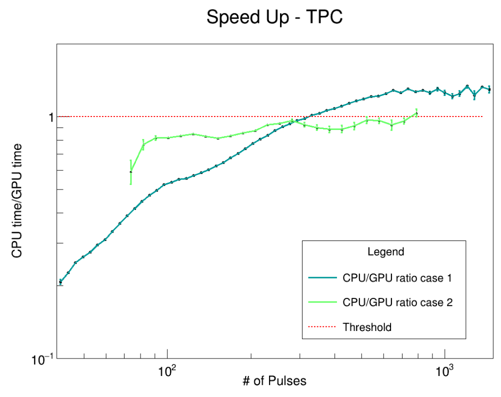
Regarding the OD, the average length of the pulses is still small (15-30 elements or 150-300 ns) but the number of pulses per waveform is generally the largest of all three detectors, reaching up to 2000 pulses per waveform. Fig. 5.8 shows the results of the performed tests for the OD. By analyzing the kernel performance only (Figs. 5.8b and 5.8d), it is possible to see that a performance gain ($\sim$1.3$\times$) is obtained by the GPU algorithm, for OD waveforms with more than $\sim$300 pulses. However, taking into account the DtoH memory transfers, the overall performance of the GPU



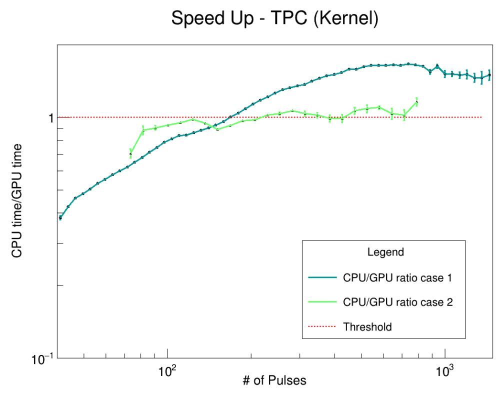
(a)



(b)



(c)



(d)

Figure 5.9: Plots showing the performance of the pulse parameterization algorithm for TPC pulses. Figures a) and b) show the processing time as a function of the number of parameterized pulses for the full GPU algorithm and for the GPU kernel respectively, when compared to the full CPU algorithm. Case 1 relates to the TPC waveforms where all pulses have roughly the same length. Case 2 relates to the TPC waveforms where pulses much larger than the average exist. Figures c) and d) show the ratio between the curves in plots a) and b). The red dashed line in these plots represents the threshold above which the GPU algorithm is advantageous performance-wise.

matches that of the CPU(Figs. 5.8a and 5.8c).

The waveforms coming from the TPC have a number of pulses that can range from 20 to 2000 pulses per waveform. Because the TPC detector can acquire S1 and S2 signals, its waveforms can have pulses whose length is much larger than the average length of the rest of the pulses in the waveform. Taking this into consideration, the analysis of the pulse parameterization of the TPC was divided into two cases: 1) waveforms where all the pulses have roughly the same length, generally ranging from 15 to 100 elements (150-1000 ns) and 2) waveforms where pulses much larger than the average length exist. The threshold used to define the length of a large pulse was set at 5000 elements (50000 ns). Both cases are presented in the plots of Fig. 5.9. Starting by case 1, Figs. 5.9b and 5.9d show that the GPU kernel starts showing performance advantages for waveforms with more than 200 pulses, with performance gains between 1.6 and 1.8×. When the DtoH memory transfers are factored in (Figs. 5.9a and 5.9c), the performance gain decreases to about 1.2-1.4× and it only starts being visible for waveforms with more than 300 pulses.

For case 2, Figs. 5.9b and 5.9d shows a performance loss compared to the CPU for waveforms with less than 200 pulses and a slight performance gain for waveforms with more. Figs. 5.9a and 5.9c show a performance loss independently of the number of pulses.

5.5 Position Reconstruction

As was previously referred in Section 4.5.2, each reconstructed position has auxiliary values associated with it. These values are: the reconstruction χ^2, the estimated energy, and the area of the reconstructed pulse. The mentioned quantities were used in this analysis to determine: a)which pulses corresponded to S2 signals (based on pulse area) and b)which of those pulses were correctly reconstructed (based on χ^2).

Since the expected results vary with the type of acquisition run, the analysis is done separately for each of the used sets of data.

DD source

The reconstruction of the DD data is expected to have a maximum of interactions near the entry point of the neutrons in the detector since the probability of interaction decays exponentially with the traveled distance.

Firstly, a cut on the pulse area is applied to select only the S2 pulses. The threshold for this cut is set at 5000 phe. Pulses with areas smaller than this are discarded from the analysis as they come mostly from S1 signals, after pulsing, single electrons, etc.

The χ^2 distribution for the pulses that satisfy this cut is presented in Fig. 5.10a. Fig. 5.10b shows the variation of χ^2 with the reconstructed position. Based on these plots, the cut for χ^2 was set at $\chi^2 < 5$.

Fig. 5.11 shows the spectrum of the estimated energy for the reconstructed interactions. A cut for $0.005 < E < 0.2\, a.u.$ was applied. The energy of the reconstructed events is calculated considering that the average energy of a ^{83m}Kr event, happening at the top of the TPC, is 1 a.u. Fig.5.12 shows a heatmap of the reconstructed positions. A hotspot of interactions can be seen near the entry point of

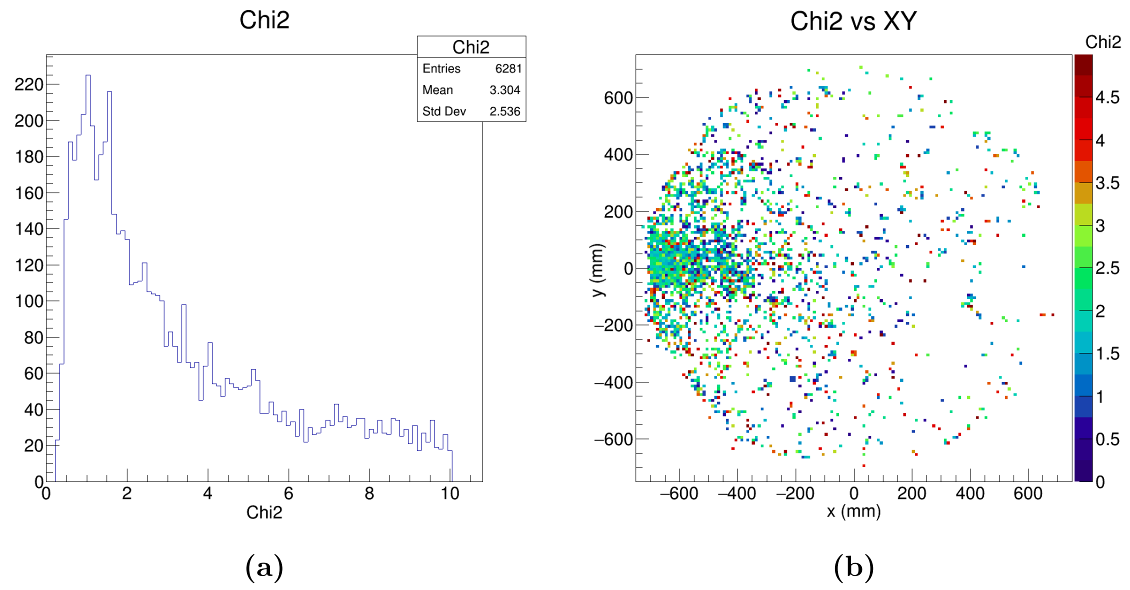


Figure 5.10: Plots showing the χ^2 distribution of the reconstructed DD data. Figure a) shows an histogram of the χ^2 distribution. Figure b) represents the χ^2 as a function of the XY position of the event.

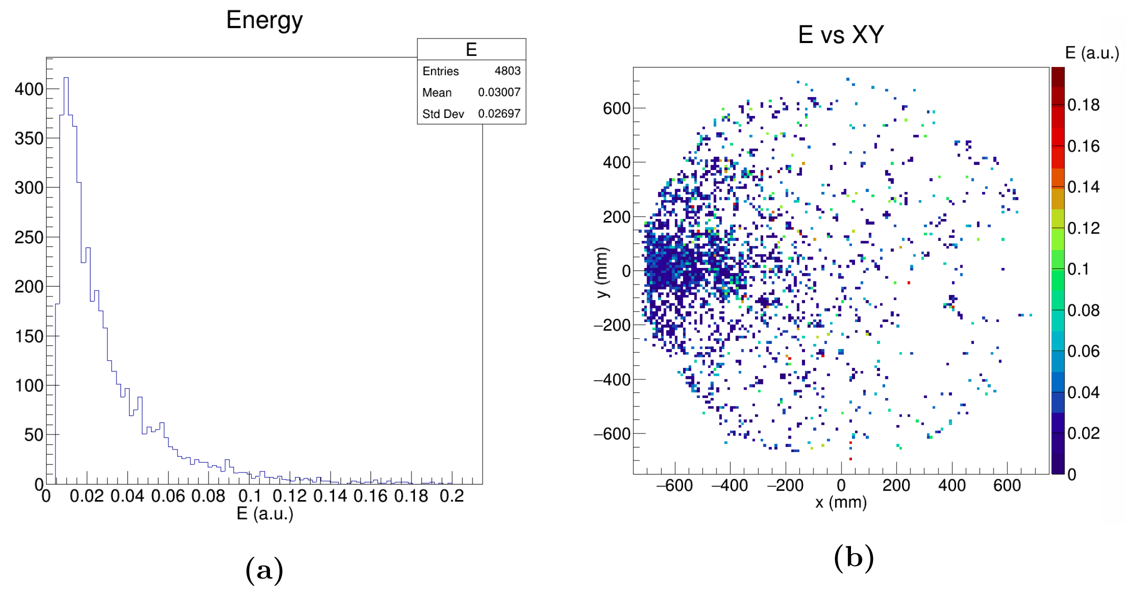


Figure 5.11: Plots showing the estimated energy distribution of the reconstructed DD data. Figure a) shows an histogram of the estimated energy. Figure b) represents the estimated energy as a function of the XY position of the event.

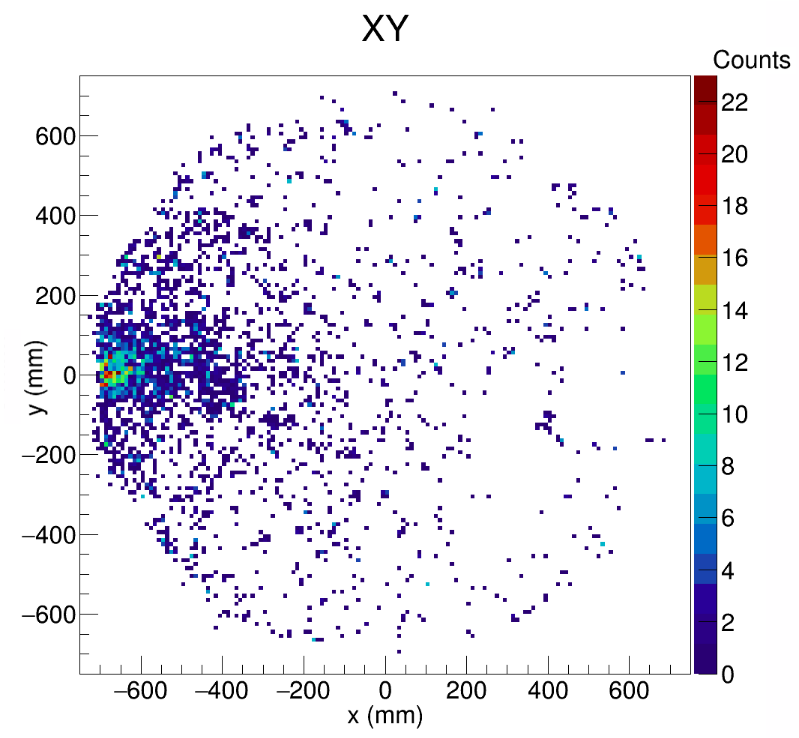
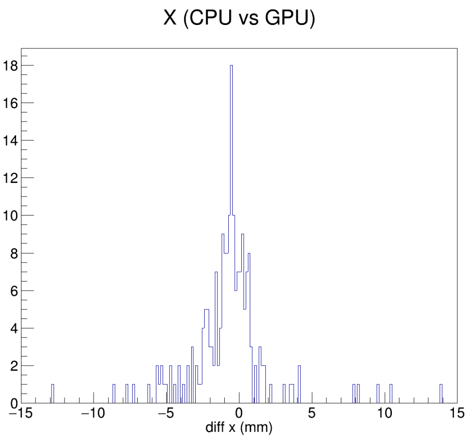


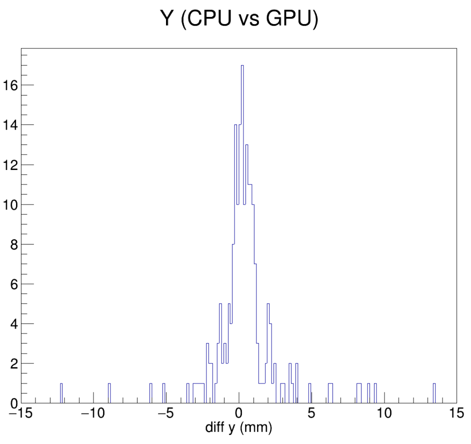
Figure 5.12: Event density of DD reconstructed events across the TPC Top PMT array. Note that this image only contains the events which satisfy all the aforementioned cuts.

the neutron beam (-750,0). The density of interactions quickly decreases with the increasing distance from the entry point, as was expected. Unfortunately, the DD statistic available to test the reconstruction was small, so the exponential decay of the density of interactions is not clearly visible, and a lot of areas appear without data in the image. Something that is also visible from Fig. 5.12 is the existence of interactions far away from the entry point of the neutron beam, which slightly contradicts what was expected. This can be explained by the fact that multiple scattering interactions are likely being reconstructed, since the selection of data based on this characteristic is not performed by the UPM (see discussion at the end of the ^{83m}Kr section). Multiple scatters can happen due to interactions of the neutrons with the detector walls or with the Skin, causing those neutrons to go off-beam, and spreading the density of interactions across the XY plane.

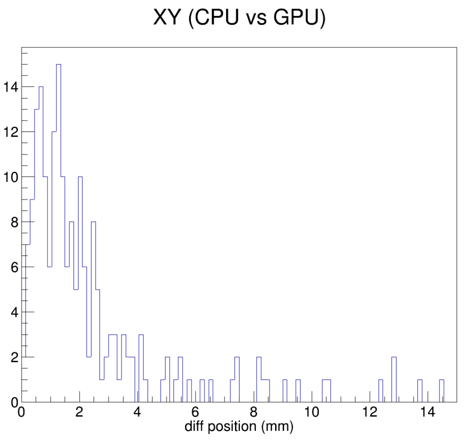
When comparing the results obtained by the CPU and the GPU, after all mentioned cuts are applied, they are within an acceptable tolerance. On the X-axis the average difference between the reconstructed positions is -0.557 ± 2.481 mm (Fig. 5.13a). On the Y-axis this average difference is of 0.266 ± 1.932 mm (Fig. 5.13b). Considering the XY plane, the average difference between the reconstructed positions is 2.192 ± 2.305 mm (Fig. 5.13c).



(a)

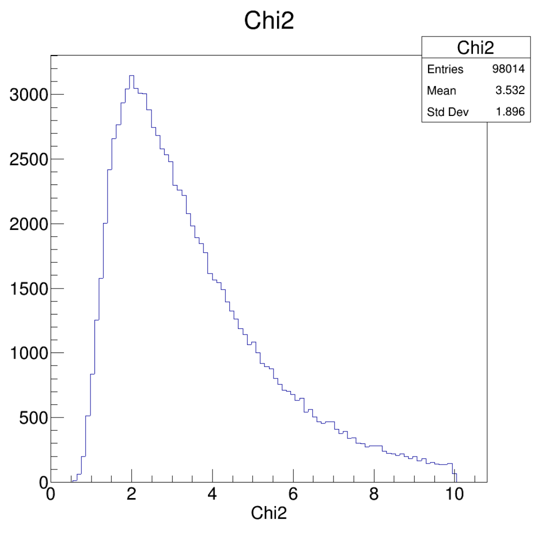


(b)

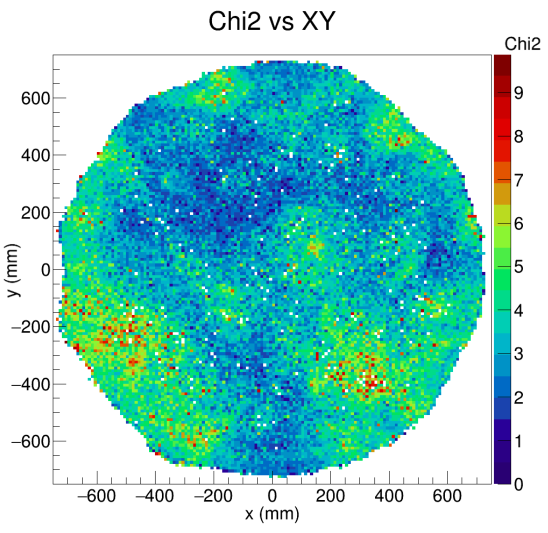


(c)

Figure 5.13: Plots showing the difference between the reconstruction results obtained by the CPU and by the GPU, for DD events.



(a)



(b)

Figure 5.14: Plots showing the χ^2 distribution of the reconstructed the raw ^{83m}Kr data. Figure a) shows an histogram of the χ^2 distribution. Figure b) represents the χ^2 as a function of the XY position of the event.

^{83m}Kr source

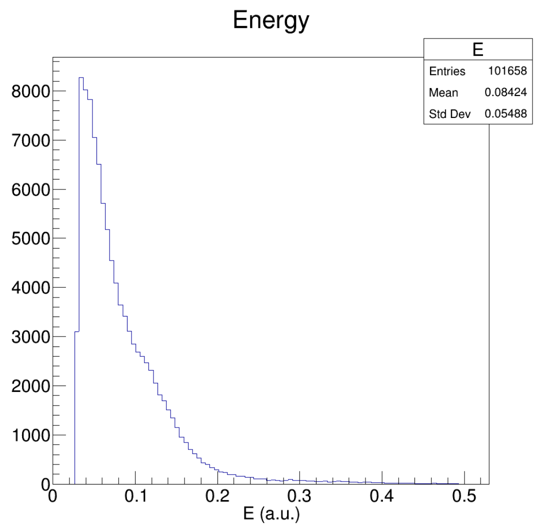
The interaction density of ^{83m}Kr events is expected to be roughly uniform across the detector. Similar to what was done for DD data, the first cut was on the pulse area. It was set at 15000 phe.

The χ^2 distribution for the pulses that satisfy this cut is presented in 5.14a. Fig. 5.14b shows the variation of χ^2 with the reconstructed position. Based on these plots, the cut for χ^2 was set at $\chi^2 < 10$.

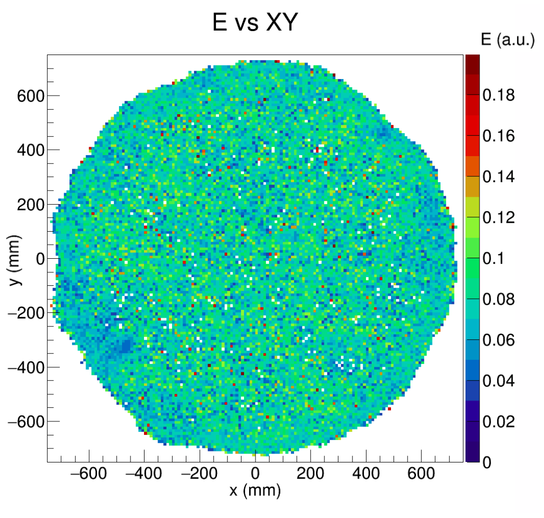
The energy spectrum is represented in Fig. 5.15. In this case a peak near 0 and a tail after 0.2 a.u. were cut. The peak near 0 a.u. corresponded to events that originated from noise sources and are thus hard to reconstruct, causing an excess of reconstructed events at the center of the detector. The heatmap of the reconstructed positions is shown in Fig. 5.16.

The comparison between the CPU and the GPU on the X-axis shows an average difference of 0.041 ± 1.157 mm (Fig. 5.17a). On the Y-axis, the average difference is -0.023 ± 1.228 mm (Fig. 5.17b). The average difference between the reconstructed positions is 1.264 ± 0.983 mm (Fig. 5.17c).

From the plots presented in Figs. 5.14b, 5.16 and 5.15b it is possible to see that the results of the reconstruction are not completely uniform, as some interaction hotspots are found in the edges of the PMT array, which generally map to regions with a higher χ^2. Furthermore, the shape of the reconstruction has some deforma-



(a)



(b)

Figure 5.15: Plots showing the energy distribution of the reconstructed raw ^{83m}Kr data. Figure a) shows an histogram of the estimated energy. Figure b) represents the estimated energy as a function of the XY position of the event.

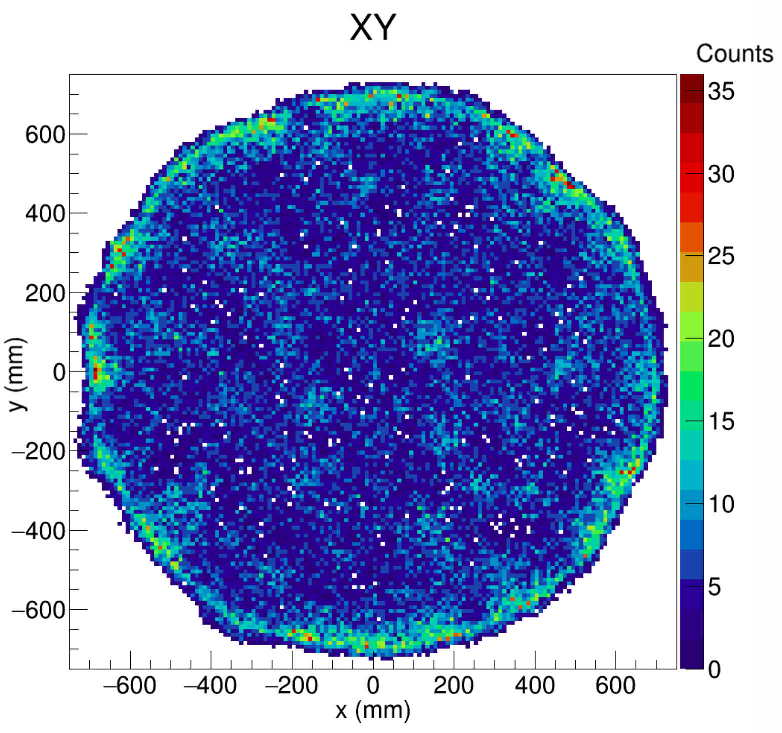
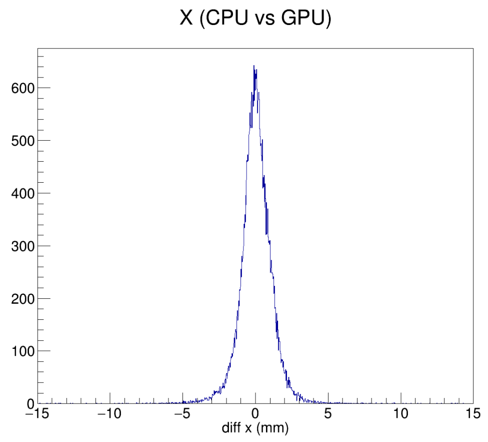
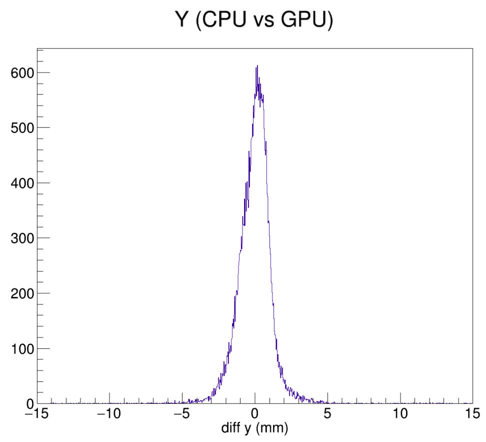


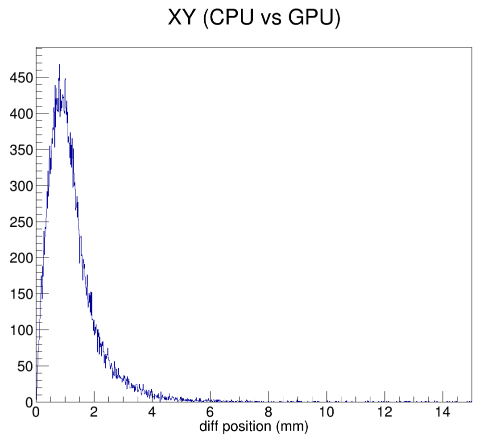
Figure 5.16: Distribution of the raw ^{83m}Kr reconstructed events across the TPC Top PMT array. Note that this image only contains the events which satisfy all the aforementioned cuts.



(a)



(b)



(c)

Figure 5.17: Plots showing the difference between the results obtained by the CPU and by the GPU, for the raw ^{83m}Kr events.

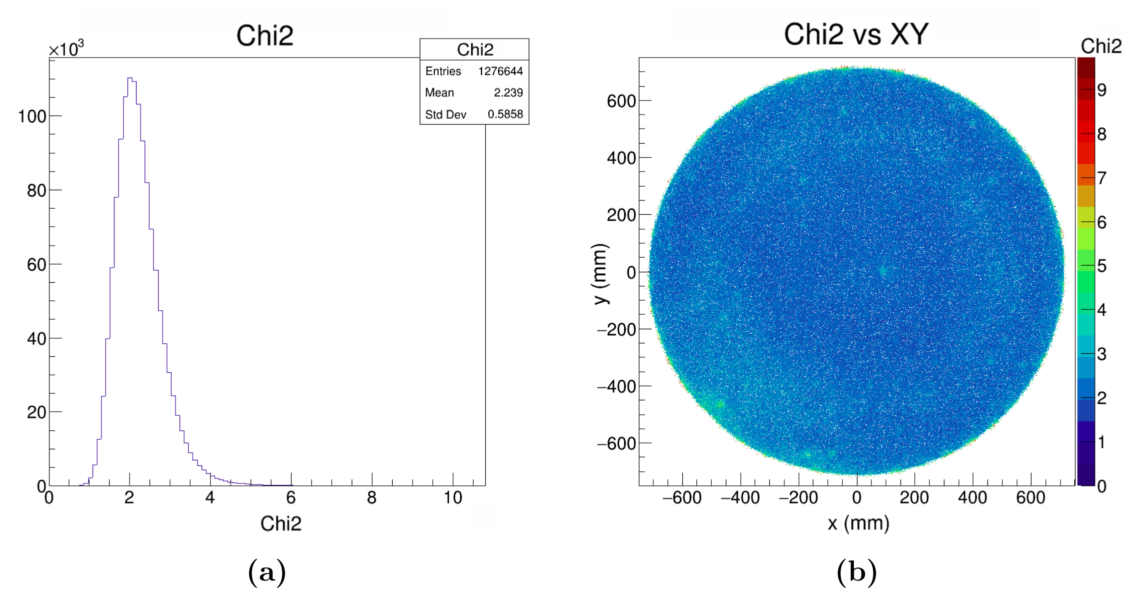


Figure 5.18: Plots showing the χ^2 distribution of the reconstructed LZap selected ^{83m}Kr data. Figure a) shows an histogram of the χ^2 distribution. Figure b) represents the χ^2 as a function of the XY position of the event.

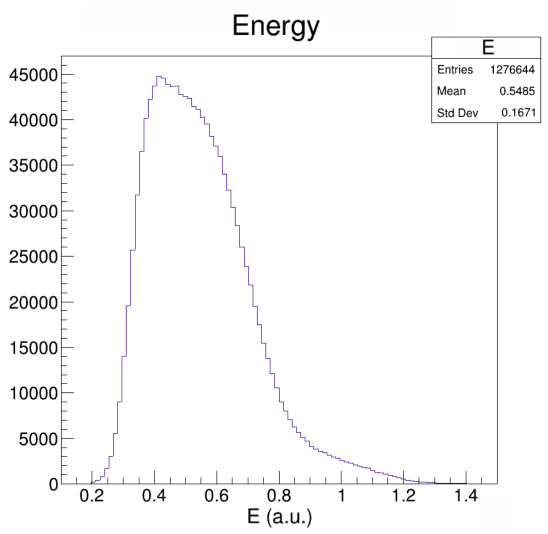
tions when it was expected to be entirely round, i.e, have the same shape as the PMT array.

To try and understand why the results of the reconstruction did not meet the expectations, a further test was conducted. This test was performed using simulated ^{83m}Kr data, processed and selected by LZap. The selected events were then injected into the reconstruction algorithm of the UPM. This data only contains S2 pulses from events that have been classified by LZap as single scatters (1 S1 + 1 S2) or "krypton-like" events (S1+S1 + S2), so there is no need to apply cuts on the pulse area.

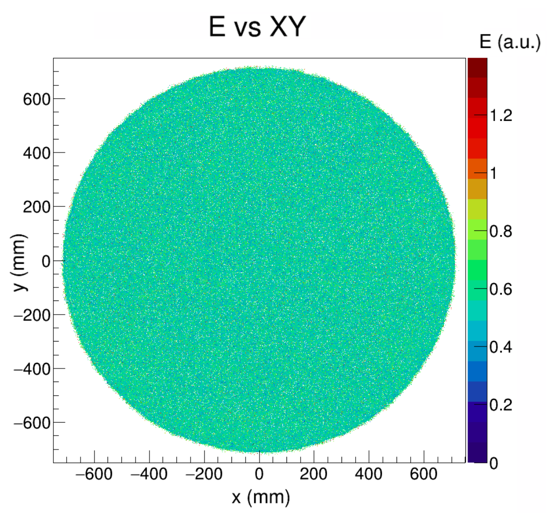
A cut for $\chi^2 < 10$ (Fig. 5.18) and $0.2 < E < 1.4\,a.u.$ (Fig. 5.19) was applied to the data. The heatmap showing the reconstructed positions is presented in Fig. 5.20.

When compared to the CPU, the average differences in the reconstructed position are: 0.001 ± 0.455 mm on the X-axis (Fig. 5.21a), 0.005 ± 0.426 mm on the Y-axis (Fig. 5.21b) and 0.579 ± 0.231 mm on the XY plane (Fig. 5.21c).

From these results it is possible to observe that the event density is uniformly distributed across the PMT array, and that the shape of the reconstruction has no deformations, as was expected. The differences between the results obtained by the raw ^{83m}Kr data (Fig. 5.16) and by the LZap ^{83m}Kr data (Fig. 5.20) can be explained by the absence of event classification and selection in the UPM. The classification process is performed in LZap by the interaction finder. The interaction finder clas-



(a)



(b)

Figure 5.19: Plots showing the energy distribution of the reconstructed LZap selected ^{83m}Kr data. Figure a) shows an histogram of the estimated energy. Figure b) represents the estimated energy as a function of the XY position of the event.

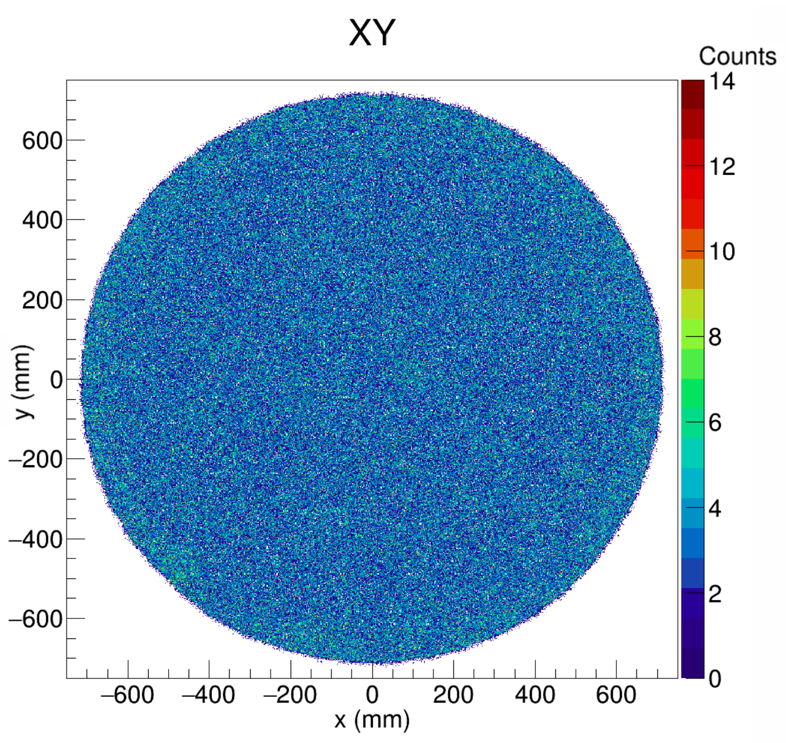


Figure 5.20: Distribution of the LZap selected ^{83m}Kr reconstructed events across the TPC Top PMT array. Note that this image only contains the events which satisfy all the aforementioned cuts.

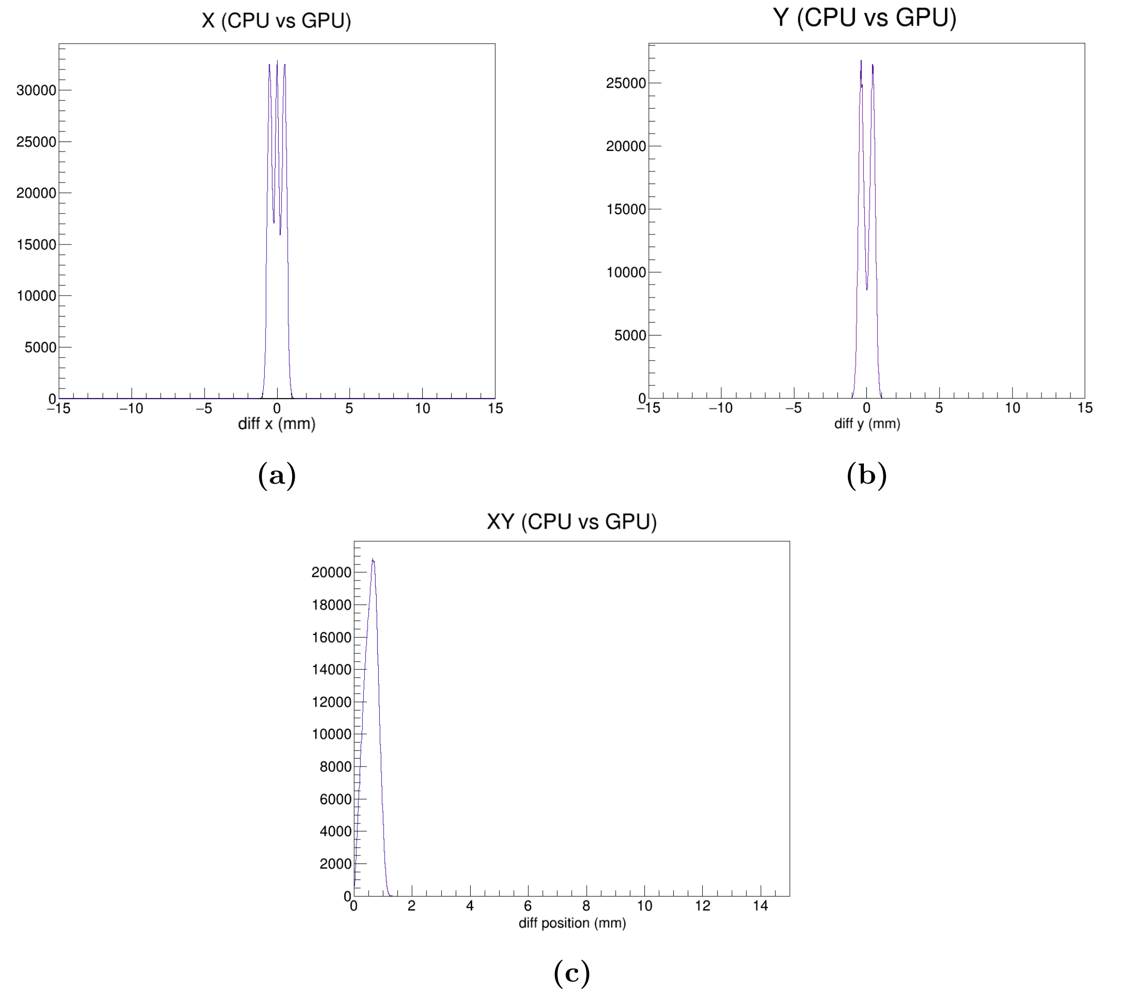


Figure 5.21: Plots showing the difference between the results obtained by the CPU and by the GPU, for the LZap selected ^{83m}Kr events.

sifies the event according to the energy deposition topology, i.e., if it corresponds to a single scatter (1 S1 + 1 S2), a double scatter (S1 + S2+S2), a multiple scatter (1 S1 + n S2), a Kr event (S1+S1 + S2), etc. The algorithm also has allowances to identify backgrounds, S2-tails, single electrons, etc. This selection process is not yet developed for the UPM, as it is not critical for the real-time analysis. More importantly, the selection of the data can never be the same for UPM and LZap. The tuning of the parameters used by the interaction finder, and other algorithms, takes years of analysis of simulated data. Since the UPM needs to be running, in real-time, at the beginning of the LZ data acquisition, these parameters will still not be fine-tuned.

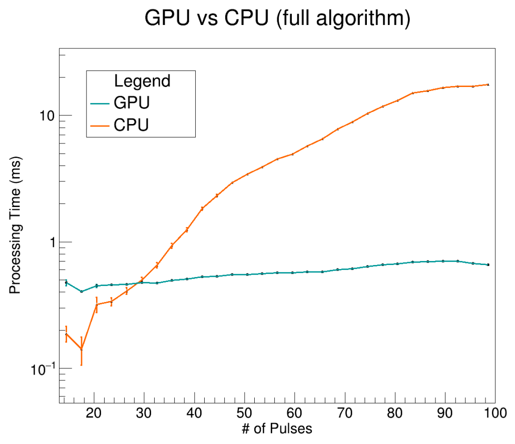
This absence of selection based on event topology on the UPM leads to the reconstruction of events with varied topologies, causing a degradation on the reconstruction as can be seen in Figs. 5.12 and 5.16.

As for the performance tests, Figs. 5.22a and 5.22c show the comparison between the processing times of the full CPU and GPU algorithms. A performance gain can be observed when more than 30 pulses are being reconstructed. That performance gap increases with the number of pulses being reconstructed, reaching up to 30× for 100 pulses. Another aspect that can be concluded from these plots is that the processing time of the GPU algorithm remains roughly constant, and thus is independent of the number of pulses being reconstructed.

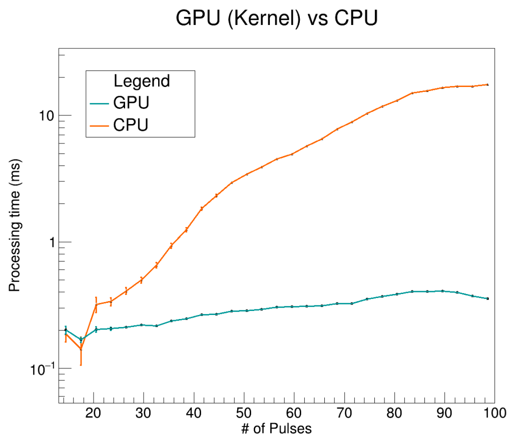
Figs. 5.22b and 5.22d show the comparison between the processing times of the full CPU algorithm and the GPU kernel. In this case, the performance gap starts at 20 reconstructed pulses and the maximum performance gain reached is $\sim$50×.

The difference between the GPU curves in Figs. 5.22a and 5.22b is explained by the time it takes to allocate and transfer the memory to the GPU. That time remains roughly constant at 0.3 ms. Contrarily to the previous algorithms the time spent on these tasks does not affect the overall performance, because the amount of data that needs to be transferred is small.

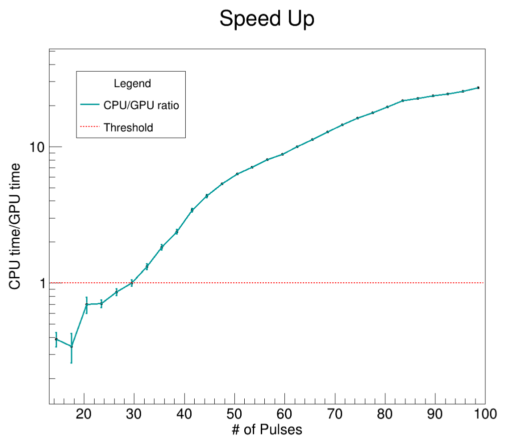
The performance differences visible in these plots is expected to increase drastically with the number of pulses. For example, the reconstruction of the LZap selected ^{83m}Kr data, which contained roughly 1.3×10^6 pulses, took just 2.51 s on the GPU, while on the CPU it took close to 6 minutes, a performance difference of about 143×. This last result is not entirely relevant for the UPM because, as its analysis is intended to be in real-time and oriented towards detector performance, the number of pulses being reconstructed will never be anywhere close to 10^6. However, it is a very interesting results for the offline analysis performed by LZap.



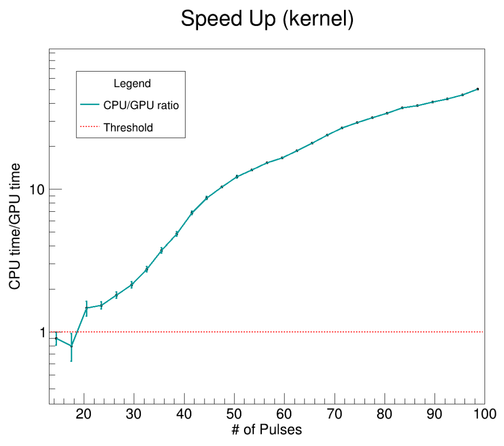
(a)



(b)



(c)



(d)

Figure 5.22: Performance of the position reconstruction algorithm. Figure a) compares the processing time of the full CPU and GPU algorithms. Figure b) compares the time spent on the GPU kernel against the full CPU algorithm. Figures c) and d) show the ratio between the CPU and GPU curves on figures a) and b), respectively.

6

Conclusions

LUX-ZEPLIN is at the forefront of experiments aiming at the direct detection of WIMPs. The combination of high sensitive mass, careful material selection and its three detectors (TPC, Skin and OD) will allow to reach unprecedented sensitivity limits for spin independent WIMP-nucleon interactions. To monitor in real-time the performance of the detectors, the UPM will be installed onsite. As of now the UPM code is developed for CPU and already running underground at SURF. However, the real-time requirement is not reached during calibration runs, using only CPUs. To meet this requirement it was proposed to use GPU-acceleration on the UPM. GPUs are an interesting option when the algorithms are computationally intensive and highly parallelizable. However, programming GPUs comes with the additional cost of the time required to allocate memory on the GPU and transfer it between CPU and GPU so, the situations where GPU-acceleration can be effectively implemented need to be carefully studied.

The objective of this project was to develop a GPU-based analysis framework to be integrated into the LZ UPM subsystem. The algorithms targeted to be part of this GPU framework were: waveform normalization, waveform sum, pulse finding, pulse parameterization and position reconstruction. All these algorithms already had a CPU version implemented on the UPM. Unfortunately, due to lack of time, the pulse finder was not developed during this project. All the other mentioned algorithms were successfully implemented on the GPU

The method chosen for the implementation of the waveform normalization was to assign one GPU thread to perform the normalization of one POD. The HtoD and DtoH memory transfers can be performed synchronously or asynchronously. The asynchronous option is used if the GPU board supports pinning previously allocated memory. For the waveform sum, a similar approach is used, where one thread sums the elements of one POD. As multiple threads might need to access the same memory address simultaneously, atomic operations are used in order to eliminate race conditions between threads.

The numerical precision tests performed for the waveform normalization and sum algorithms show identical results to the ones obtained by the CPU version, within the standard floating point precision. A performance gain varying from $1.1\times$ to $1.4\times$ is observed for events with sizes between $2{\times}10^6$ and $5{\times}10^6$ elements. If only the time spent on the GPU kernel is considered, a performance gain between 3 and $4\times$ is observed, and it remains roughly constant for all event sizes. This last result confirms that the main limitation for this algorithm is the time spent on allocating and transferring the memory to the GPU.

The pulse parameterization algorithm was divided into two steps. The first step calculates the areas, cumulative integrals and the TBA. In this step, the pulse is divided by the launched GPU threads, and each thread calculates the mentioned quantities for their portion of the pulse. The second step calculates the fixed area times and fixed area fractions. For this step, one launched GPU thread calculates the mentioned quantities for one pulse.

The numerical precision tests show once again equal results to the CPU within the standard floating point precision. No performance advantage is verified when the pulses being parameterized are from the Skin or OD detectors. A performance gain of approximately 1.2 to $1.4\times$ is observed for TPC events with more than 300 pulses, in the case where all pulses in the event have similar sizes. If only the kernel processing time is considered, the performance gain for the TPC pulses goes up to 1.6-$1.8\times$ and starts becoming apparent for events with more than 200 pulses. A small performance gain of approximately $1.3\times$ is achieved in this case for OD pulses.

Finally, the position reconstruction algorithm was ported from the ANTS software package [69] [70]. It uses a "contracting grids" method to reconstruct the position of the events. Unlike the previous algorithms this one is only applied to data coming from the TPC Top PMT array, since only the S2 pulses are to be reconstructed. An estimation of the event energy and the reconstruction χ^2 are also calculated. These values are combined with the area of the pulse to check the quality of the reconstruction, and select the pulses which have been correctly reconstructed.

The CPU code used to compare the results of the numerical tests is a standalone version of the Mercury reconstruction module for LZap [67] [68].

The position reconstruction was tested with three sets of data. The reconstruction of raw DD events has an average difference of -0.557 ± 2.481 mm on the X-axis, when compared to the CPU results. On the Y-axis the difference is 0.266 ± 1.932 mm and on the XY plane is 2.192 ± 2.305 mm. The cuts applied to these events were $A > 5000\,phe$, $0.005 < E < 0.2\,a.u.$ and $\chi^2 < 5$.

With raw ^{83m}Kr events the average difference is 0.041 ± 1.157 mm on the X-axis,

-0.023 ± 1.228 mm on the Y-axis and 1.264 ± 0.983 mm on the XY plane. The cuts applied were $A > 15000\,phe$, $0.03 < E < 0.2\,a.u.$ and $\chi^2 < 10$.

With the ^{83m}Kr events selected with LZap the average difference was 0.001 ± 0.455 mm on the X-axis, 0.005 ± 0.426 mm on the Y-axis and 0.579 ± 0.231 mm on the XY plane. The cuts applied were $0.2 < E < 1.4\,a.u.$ and $\chi^2 < 10$.

A performance advantage is achieved by the GPU implementation when more than 30 pulses are being reconstructed. This performance advantage reaches $\sim 30\times$ when more than 100 pulses are being reconstructed. If only the kernel time is considered, the performance advantage starts being noticeable for more than 20 pulses and it reaches $\sim 50\times$ for 100 reconstructed pulses. When the amount of pulses being reconstructed reaches 10^6 the performance advantage of the GPU reaches $143\times$. This final value can be particularly interesting for the analysis performed by LZap.

The characterization of the algorithms developed during this work has been able to give valuable insight about the use of GPUs within the UPM. More importantly, it allowed to identify performance bottlenecks, namely related to memory transfers, which shall be addressed as the development of the UPM GPU-framework continues and more complex algorithms are implemented on the GPU.

6.1 Future Work

The immediate future of this project is the development of the pulse finder. After this, the GPU analysis framework is complete and ready to be implemented into the UPM version already running underground at SURF.

Studying possible improvements to the performance of the already developed GPU algorithms is something that is going to be looked at. One of the major limitations found during this project is related to the time consumed by the memory transfer operations. A possible solution that has been considered is performing the analysis on the GPU per event instead of per algorithm, i.e., when an event is sent to the GPU-framework its analysis is performed entirely on the GPU. This solution would allow to optimize memory transfers, which would only be performed at two separate occasions (beginning and end of the analysis). Furthermore, the kernels can be modified to better work as a whole instead of separate algorithms, further improving the overall performance.

Another solution that should be tested is the use of unified memory. This is a type of memory that allows the CPU and the GPU to access the same memory address, which allows to increase the bandwidth of the memory transfers. This feature is only fully supported by GPUs based on Pascal architecture or better [75]. Unfortunately, the GPU board used in this work is based on the Maxwell architecture (previous to Pascal) and does not fully support this feature.

Adapting the algorithms into a multi-GPU system is also going to be considered. This could help enhance the performance gains already observed in some of the algorithms. For this, an efficient way for CPU/GPU and GPU/GPU communications to occur needs to be explored.

The development of smart decision structures, using Machine Learning on GPUs, is also a possibility of future improvement for the UPM. These structures would allow automatic detection of possible performance issues on the detectors, and action the relevant LZ control mechanisms. This was part of the initial plan of this project and could not be performed due to lack of time.

LZap analysis can also benefit from GPU-acceleration. This is especially true for the position reconstruction algorithm, whose performance gain reached approximately 143$\times$. The gains obtained by the remaining algorithms were minimal and would potentially not be present in a multi-CPU system.

Bibliography

[1] D. N. M. and, “The LZ dark matter experiment,” *Journal of Physics: Conference Series*, vol. 718, p. 042039, may 2016.

[2] B. J. Mount, S. Hans, R. Rosero, M. Yeh, C. Chan, R. J. Gaitskell, D. Q. Huang, J. Makkinje, D. C. Malling, M. Pangilinan, C. A. Rhyne, W. C. Taylor, and et. al, “LUX-ZEPLIN (LZ) technical design report,” 2017.

[3] G. Jungman, M. Kamionkowski, and K. Griest, “Supersymmetric dark matter,” *Physics Reports*, vol. 267, no. 5, pp. 195 – 373, 1996.

[4] V. C. Rubin, “Dark matter in spiral galaxies,” *Scientific American*, vol. 248, no. 6, pp. 96–109, 1983.

[5] P. Brás, “Finding a needle in a haystack: Background studies & WIMP detection efficiency in LUX,” 2015.

[6] “Dark matter in galaxies and clusters.” https://ned.ipac.caltech.edu/. Accessed: 2020-07-10.

[7] K. A. Olive, “Review of particle physics,” *Chinese Physics. C, High Energy Physics and Nuclear Physics*, vol. 40, 10 2016.

[8] T.-C. Chang, U.-L. Pen, J. B. Peterson, and P. McDonald, “Baryon acoustic oscillation intensity mapping of dark energy,” *Phys. Rev. Lett.*, vol. 100, p. 091303, Mar 2008.

[9] A. Slosar, V. Iršič, D. Kirkby, S. Bailey, and et al., “Measurement of baryon acoustic oscillations in the lyman-α forest fluctuations in BOSS data release 9,” *Journal of Cosmology and Astroparticle Physics*, vol. 2013, p. 026–026, Apr 2013.

[10] “What is the acoustic peak?.” https://www.cfa.harvard.edu/. Accessed: 2020-07-05.

[11] J. Beringer, J. F. Arguin, R. M. Barnett, K. Copic, O. Dahl, D. E. Groom, C. J. Lin, J. Lys, H. Murayama, C. G. Wohl, W. M. Yao, P. A. Zyla, C. Amsler, M. Antonelli, and et al., "Review of particle physics," *Phys. Rev. D*, vol. 86, p. 010001, Jul 2012.

[12] R. C. V. Coelho, M. O. Calvão, R. R. R. Reis, and B. B. Siffert, "Standardization of type Ia supernovae," *European Journal of Physics*, vol. 36, p. 015007, nov 2014.

[13] W. Hillebrandt, M. Kromer, F. K. Röpke, and A. J. Ruiter, "Towards an understanding of type ia supernovae from a synthesis of theory and observations," *Frontiers of Physics*, vol. 8, apr 2013.

[14] D. Clowe, M. Bradač, A. H. Gonzalez, M. Markevitch, S. W. Randall, C. Jones, and D. Zaritsky, "A direct empirical proof of the existence of dark matter," *The Astrophysical Journal*, vol. 648, pp. L109–L113, aug 2006.

[15] N. Aghanim, Y. Akrami, M. Ashdown, J. Aumont, C. Baccigalupi, M. Ballardini, A. J. Banday, R. B. Barreiro, N. Bartolo, and et al., "Planck 2018 results," *Astronomy & Astrophysics*, vol. 641, p. A6, Sep 2020.

[16] "Legacy Archive for Microwave Background Data Analysis." https://lambda.gsfc.nasa.gov/. Accessed: 2020-07-12.

[17] M. Tanabashi, P. Grp, K. Hagiwara, K. Hikasa, K. Nakamura, Y. Sumino, F. Takahashi, J. Tanaka, K. Agashe, G. Aielli, C. Amsler, M. Antonelli, D. Asner, H. Baer, S. Banerjee, R. Barnett, T. Basaglia, C. Bauer, and J. Beatty, "Review of particle physics: Particle data group," *Physical Review D*, vol. 98, 08 2018.

[18] R. D. Peccei, "The strong CP problem and axions," *Axions*, p. 3–17, 2008.

[19] A. Derevianko, V. A. Dzuba, V. V. Flambaum, and M. Pospelov, "Axio-electric effect," *Physical Review D*, vol. 82, Sep 2010.

[20] T. Braine, R. Cervantes, N. Crisosto, N. Du, S. Kimes, L. J. Rosenberg, G. Rybka, J. Yang, D. Bowring, A. S. Chou, R. Khatiwada, A. Sonnenschein, and et al., "Extended search for the invisible axion with the axion dark matter experiment," *Phys. Rev. Lett.*, vol. 124, p. 101303, Mar 2020.

[21] E. Aprile, J. Aalbers, F. Agostini, M. Alfonsi, L. Althueser, F. D. Amaro, V. C. Antochi, E. Angelino, J. R. Angevaare, F. Arneodo, D. Barge, L. Baudis,

B. Bauermeister, L. Bellagamba, and et al., “Observation of excess electronic recoil events in XENON1T,” 2020.

[22] M. Szydagis, C. Levy, G. M. Blockinger, A. Kamaha, N. Parveen, and G. R. C. Rischbieter, “Investigating the xenon1t low-energy electronic recoil excess using nest,” 2020.

[23] Y. Sofue, “Grand rotation curve and dark-matter halo in the Milky Way galaxy,” *Publications of the Astronomical Society of Japan*, vol. 64, 08 2012. 75.

[24] L. E. Strigari, “Dark matter in dwarf spheroidal galaxies and indirect detection: a review,” *Reports on Progress in Physics*, vol. 81, p. 056901, mar 2018.

[25] P. Cushman, C. Galbiati, D. N. McKinsey, H. Robertson, T. M. P. Tait, D. Bauer, A. Borgland, and et al., “Snowmass CF1 summary: WIMP dark matter direct detection,” 2013.

[26] A. A. Petrov and W. Shepherd, “Searching for dark matter at lhc with mono-higgs production,” *Physics Letters B*, vol. 730, pp. 178 – 183, 2014.

[27] S. Haselschwardt, S. Shaw, H. Nelson, M. Witherell, M. Yeh, K. Lesko, A. Cole, S. Kyre, and D. White, “A liquid scintillation detector for radioassay of gadolinium-loaded liquid scintillator for the LZ outer detector,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 937, p. 148–163, Sep 2019.

[28] “PICO collaboration website.” http://www.picoexperiment.com/. Accessed: 2020-09-22.

[29] R. Bernabei, P. Belli, A. Bussolotti, F. Cappella, V. Caracciolo, R. Cerulli, C.-J. Dai, A. D’Angelo, A. Di Marco, H.-L. He, and et al., “First model independent results from DAMA/LIBRA–Phase2,” *Universe*, vol. 4, p. 116, Nov 2018.

[30] R. Agnese, A. J. Anderson, T. Aramaki, M. Asai, W. Baker, D. Balakishiyeva, D. Barker, R. Basu Thakur, and et al., “New results from the search for low-mass Weakly Interacting Massive Particles with the CDMS low ionization threshold experiment,” *Phys. Rev. Lett.*, vol. 116, p. 071301, Feb 2016.

[31] J. Battat, J. Brack, E. Daw, A. Dorofeev, A. Ezeribe, J.-L. Gauvreau, M. Gold, J. Harton, J. Landers, and et al., “First background-free limit from a directional dark matter experiment: Results from a fully fiducialised DRIFT detector,” *Physics of the Dark Universe*, vol. 9-10, pp. 1 – 7, 2015.

[32] V. Chepel and H. Araújo, "Liquid noble gas detectors for low energy particle physics," *Journal of Instrumentation*, vol. 8, p. R04001–R04001, Apr 2013.

[33] X. Cui, A. Abdukerim, W. Chen, X. Chen, Y. Chen, B. Dong, D. Fang, C. Fu, K. Giboni, F. Giuliani, and et al., "Dark matter results from 54-ton-day exposure of PandaX-II experiment," *Phys. Rev. Lett.*, vol. 119, p. 181302, Oct 2017.

[34] D. N. McKinsey, D. Akerib, S. Bedikian, A. Bernstein, A. Bolozdynya, A. Bradley, J. Chapman, and et al., "The LUX dark matter search," *Journal of Physics: Conference Series*, vol. 203, p. 012026, jan 2010.

[35] E. Aprile, J. Aalbers, F. Agostini, M. Alfonsi, F. D. Amaro, M. Anthony, B. Antunes, F. Arneodo, M. Balata, and et al., "The XENON1T dark matter experiment," *The European Physical Journal C*, vol. 77, Dec 2017.

[36] E. Aprile, J. Aalbers, F. Agostini, M. Alfonsi, L. Althueser, F. Amaro, M. Anthony, F. Arneodo, L. Baudis, B. Bauermeister, and et al., "Dark matter search results from a one ton-year exposure of XENON1T," *Physical Review Letters*, vol. 121, Sep 2018.

[37] R. Ajaj, P.-A. Amaudruz, G. R. Araujo, M. Baldwin, M. Batygov, B. Beltran, C. E. Bina, and et al., "Search for dark matter with a 231-day exposure of liquid argon using DEAP-3600 at SNOLAB," *Phys. Rev. D*, vol. 100, p. 022004, Jul 2019.

[38] C. R. and, "The argon dark matter experiment (ArDM)," *Journal of Physics: Conference Series*, vol. 203, p. 012024, jan 2010.

[39] J. N. Marx and D. R. Nygren, "The Time Projection Chamber," *Phys. Today*, vol. 31N10, pp. 46–53, 1978.

[40] V. Lepeltier, "Review on TPCs," *Journal of Physics: Conference Series*, vol. 65, p. 012001, apr 2007.

[41] M. Schumann, "Dual-phase liquid xenon detectors for dark matter searches," *Journal of Instrumentation*, vol. 9, p. C08004–C08004, Aug 2014.

[42] "LUX collaboration website." http://luxdarkmatter.org/. Accessed: 2020-07-10.

[43] D. Akerib, S. Alsum, H. Araújo, X. Bai, A. Bailey, J. Balajthy, P. Beltrame, E. Bernard, A. Bernstein, T. Biesiadzinski, and et al., "Results from a search for

dark matter in the complete LUX exposure," *Physical Review Letters*, vol. 118, Jan 2017.

[44] D. Akerib, H. Araújo, X. Bai, A. Bailey, J. Balajthy, S. Bedikian, E. Bernard, A. Bernstein, A. Bolozdynya, A. Bradley, and et al., "First results from the LUX dark matter experiment at the sanford underground research facility," *Physical Review Letters*, vol. 112, Mar 2014.

[45] E. Aprile, J. Aalbers, F. Agostini, M. Alfonsi, F. Amaro, M. Anthony, F. Arneodo, P. Barrow, L. Baudis, B. Bauermeister, and et al., "First dark matter search results from the XENON1T experiment," *Physical Review Letters*, vol. 119, Oct 2017.

[46] A. Brown, "The future of dark matter search with XENONnT," 2019.

[47] N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming.* Addison-Wesley, 2013.

[48] M. Harris, "Using shared memory in CUDA C/C++." from NVIDIA Developer Blog.

[49] M. Harris, "How to optimize data transfers in CUDA C/C++." from NVIDIA Developer Blog.

[50] M. Harris, "How to overlap data transfers in CUDA C/C++." from NVIDIA Developer Blog.

[51] G. Pratx and L. Xing, "Gpu computing in medical physics: A review," *Medical physics*, vol. 38, pp. 2685–97, 05 2011.

[52] X. Jia, P. Ziegenhein, and S. B. Jiang, "GPU-based high-performance computing for radiation therapy," *Physics in Medicine and Biology*, vol. 59, pp. R151–R182, feb 2014.

[53] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," p. 91–98, 1994.

[54] Y. Okitsu, F. Ino, and K. Hagihara, "High-performance cone beam reconstruction using CUDA compatible GPUs," *Parallel Computing*, vol. 36, no. 2, pp. 129 – 141, 2010.

[55] P. Noël, A. Walczak, J. Xu, J. Corso, K. Hoffmann, and S. Schafer, "Gpu-based cone beam computed tomography," *Computer methods and programs in biomedicine*, vol. 98, pp. 271–7, 09 2009.

[56] X. Jia, X. Gu, J. Sempau, D. Choi, A. Majumdar, and S. B. Jiang, "Development of a GPU-based monte carlo dose calculation code for coupled electron–photon transport," *Physics in Medicine and Biology*, vol. 55, pp. 3077–3086, may 2010.

[57] X. Jia, X. Gu, Y. Graves, M. Folkerts, and S. Jiang, "GPU-based fast monte carlo simulation for radiotherapy dose calculation," *Physics in medicine and biology*, vol. 56, pp. 7017–31, 11 2011.

[58] C. F. Nielsen, "GPU accelerated simulation of channeling radiation of relativistic particles," 2019.

[59] B. S. C, "Opticks : GPU optical photon simulation for particle physics using NVIDIA® OptiX^{TM}," *Journal of Physics: Conference Series*, vol. 898, p. 042001, oct 2017.

[60] K. Lim, Y. Hong, Y. Choi, and H. Byun, "Real-time traffic sign recognition based on a general purpose GPU and deep-learning," *PLoS One*, vol. 12, mar 2017.

[61] V. Prisacariu and I. Reid, "fastHOG - a real-time GPU implementation of HOG," no. 2310/09, 2009.

[62] M. Vogelgesang, S. Chilingaryan, T. d. Rolo, and A. Kopmann, "Ufo: A scalable gpu-based image processing framework for on-line monitoring," pp. 824–829, 2012.

[63] Y. Fang, Q. Chen, and N. Xiong, "A multi-factor monitoring fault tolerance model based on a GPU cluster for big data processing," *Information Sciences*, vol. 496, pp. 300 – 316, 2019.

[64] A. P. Santhanam, Y. Min, H. Neelakkantan, N. Papp, S. L. Meeks, and P. A. Kupelian, "A multi-GPU real-time dose simulation software framework for lung radiotherapy," *International Journal of Computer Assisted Radiology and Surgery*, vol. 7, 2012.

[65] A. Badal, F. Zafar, H. Dong, and A. Badano, "A real-time radiation dose monitoring system for patients and staff during interventional fluoroscopy using a GPU-accelerated Monte Carlo simulator and an automatic 3D localization system based on a depth camera," 2013.

[66] M. M. Rathore, H. Son, A. Ahmad, and A. Paul, "Real-time video processing for traffic control in smart city using Hadoop ecosystem with GPUs," 2018.

[67] V. N. Solovov, V. A. Belov, D. Y. Akimov, H. M. Araujo, E. J. Barnes, A. A. Burenkov, V. Chepel, A. Currie, L. DeViveiros, B. Edwards, and et al., "Position reconstruction in a dual phase xenon scintillation detector," *IEEE Transactions on Nuclear Science*, vol. 59, p. 3286–3293, Dec 2012.

[68] Mercury GitHub repository: https://github.com/vovasolo/reconstructor.

[69] A. Morozov, V. Solovov, R. Martins, F. Neves, V. Domingos, and V. Chepel, "Ants2 package: simulation and experimental data processing for anger camera type detectors," *Journal of Instrumentation*, vol. 11, p. P04022–P04022, Apr 2016.

[70] ANTS2 GitHub repository: https://github.com/andrmor/ANTS2.

[71] D. Akerib, S. Alsum, H. Araújo, X. Bai, A. Bailey, J. Balajthy, P. Beltrame, E. Bernard, A. Bernstein, T. Biesiadzinski, and et al., "Position reconstruction in LUX," *Journal of Instrumentation*, vol. 13, p. P02001–P02001, Feb 2018.

[72] Intel Core i7-6700 datasheet.

[73] NVIDIA GeForce GTX 970.

[74] D. Akerib, C. Akerlof, A. Alqahtani, S. Alsum, T. Anderson, N. Angelides, H. Araújo, J. Armstrong, M. Arthurs, X. Bai, and et al., "Simulations of events for the LUX-ZEPLIN (LZ) dark matter experiment," *Astroparticle Physics*, p. 102480, Jun 2020.

[75] M. Harris, "Unified memory for cuda beginners." from NVIDIA Developer Blog.