UNIVERSIDAD
DE MÁLAGA

Escuela de Ingenierías Industriales

Programa de Doctorado en Ingeniería Mecatrónica

Departamento de
Arquitectura de Computadores

TESIS DOCTORAL

# Accelerating Massive Sensor-based Analytics

Felipe Muñoz López

November 2024

**Dirigida por:**
Prof. Rafael Asenjo Plaza
Prof. Mª Ángeles González Navarro



AUTOR: Felipe Muñoz López

https://orcid.org/0000-0001-8706-4266

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga



UNIVERSIDAD
DE MÁLAGA

# DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña FELIPE MUÑOZ LÓPEZ
Estudiante del programa de doctorado INGENIERÍA MECATRÓNICA de la Universidad de Málaga, autor/a de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: ACCELERATING MASSIVE SENSOR-BASED ANALYTICS

Realizada bajo la tutorización de RAFAEL ASENJO PLAZA y dirección de MARÍA ÁNGELES GONZÁLEZ NAVARRO Y RAFAEL ASENJO PLAZA (si tuviera varios directores deberá hacer constar el nombre de todos)

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 01 de OCTUBRE de 2024

| | |
|---|---|
| Fdo.: FELIPE MUÑOZ LÓPEZ<br>Doctorando/a | Fdo.: RAFAEL ASENJO PLAZA<br>Tutor/a (En el ejercicio de sus funciones, firma el Coordinador del Programa de Doctorado) |
| Fdo.: Mª ÁNGELES GONZÁLEZ NAVARRO Y RAFAEL ASENJO PLAZA<br>Director/es de tesis | |


Edificio Pabellón de Gobierno. Campus El Ejido. 29071
**Tel.:** 952 13 10 28 / 952 13 14 61 / 952 13 71 10
**E-mail:** doctorado@uma.es

Edificio Pabellón de Gobierno. Campus El Ejido. 29071
**Tel.:** 952 13 10 28 / 952 13 14 61 / 952 13 71 10
**E-mail:** doctorado@uma.es

Dr. Rafael Asenjo Plaza.
Catedrático del Departamento de Arquitectura de Computadores de la Universidad de Málaga.

Dra. Mª Ángeles González Navarro.
Catedrática del Departamento de Arquitectura de Computadores de la Universidad de Málaga.

**CERTIFICAN:**

Que la memoria titulada "Accelerating Massive Sensor-based Analytics ", ha sido realizada por D. Felipe Muñoz López bajo nuestra dirección en el Departamento de Arquitectura de Computadores de la Universidad de Málaga y constituye la Tesis que presenta para optar al grado de Doctor en Ingeniería Mecatrónica.

Málaga, 25 de noviembre de 2024

Prof. Mª Ángeles González Navarro.
Codirectora de la tesis.

Prof. Rafael Asenjo Plaza.
Codirector de la tesis.

(En ejercicio de sus funciones, firma el Coordinador del Programa de Doctorado)

UNIVERSIDAD
DE MÁLAGA

# Autorización para la lectura de la Tesis e Informe de la utilización de las publicaciones que la avalan

Los abajo firmantes declaran, bajo su responsabilidad, que autorizan la lectura de la tesis de la doctoranda D. Felipe Muñoz López con DNI titulada "Accelerating Massive Sensor-based Analytics " y que ninguna de las publicaciones que avalan dicha tesis han sido utilizadas en tesis anteriores.

Málaga, 25 de noviembre de 2024

Prof. Mª Ángeles González Navarro.
Codirectora de la tesis.

Prof. Rafael Asenjo Plaza.
Codirector de la tesis.
(En ejercicio de sus funciones, firma el Coordinador del Programa de Doctorado)

*Dedicada a mi familia.*

# Abstract

This thesis addresses the challenges and opportunities in the field of "massive data analytics" by developing novel methods and tools for information extraction and analysis, focusing on the intersection of information theory, algorithmic approaches, and architectural considerations in two specific sensor-based applications: ground point extraction using Light Detection and Ranging (LiDAR) technology and epileptic seizure detection using Electroencephalography (EEG). The objective of this research aims to develop novel methods and tools for the extraction and analysis of information from large-scale sensor data, emphasizing accuracy, as well as computational and power efficiencies on different architectures, ultimately deploying these methodologies on edge devices for real-time applications. We target multicore Central Processing Unit (CPU), integrated and discrete Graphics Processing Unit (GPU), as well as low-power heterogeneous System-on-Chip (SoC).

The first part of the thesis examines processing techniques associated with LiDAR data, addressing the significant challenges posed by the volume and complexity of the data generated by this technology, in particular the careful selection of the right data structure tailored to the microarchitectural features of each accelerator in a heterogeneous platform. By utilizing the Overlap Window Method (OWM) and advanced algorithmic optimizations oriented to minimize memory access and exploit all available parallelism in the corresponding accelerator, this research proposes efficient solutions for ground point extraction from LiDAR point clouds, establishing a foundation for real-time processing methodologies. The implemented code has been successfully deployed on various architectures, including dedicated Graphics Processing Unit (GPU) from NVIDIA and integrated GPU from Intel. Ongoing work aims to deploy this code on an embedded device, specifically the Jetson Nano from NVIDIA.

The second part of the thesis focusses on epileptic seizure detection through EEG analysis. It introduces a novel approach named Patterns augmented by Features Epileptic Seizure Detection (PaFESD), which combines Dynamic Time Warping (DTW), for extracting characteristic patterns in epileptic seizures, with feature-based signal analysis to enhance the accuracy of seizure identification. The research underscores the importance of architectural considerations in algorithm design, highlighting the potential for advancements in edge computing and

real-time data processing. Several works from our team, resulting from this thesis, are deploying the outputs of PaFESD on embedded devices such as Raspberry Pis and RISC-V Microcontroller Unit (MCU). Additionally, other outcomes of this thesis include efforts to accelerate stages of PaFESD using Field-Programmable Gate Array (FPGA) and GPU accelerators, further demonstrating the versatility and efficiency of the proposed methodologies.

Overall, this research is guided by interdisciplinary collaboration and leverages technological advancements in computing architectures, Application Programming Interface (API) for heterogeneous computing, and parallel processing techniques. The objectives encompass advancing LiDAR data processing techniques, enhancing epileptic seizure detection methodologies, and leveraging technological innovations based on novel programming models for several architectures to contribute to both theoretical frameworks and real-world applications in the field of massive sensor-based analytics.

# Contents







# List of Figures

# List of Tables







# Acronyms

**AI** Artificial Intelligence. 10–12, 15, 17, 19, 21, 29, 34

**AI** Arithmetic Intensity. 93, 94

**ALS** Airborne Laser Scanning. 3, 49, 50

**ALU** Arithmetic Logic Unit. 25

**AOT** Ahead-of-Time. 34, 35

**AP** Attainable Performance. 93, 94

**API** Application Programming Interface. II, 30, 35, 37, 38, 47

**AR** Augmented Reality. 11, 29

**ASIC** Application-Specific Integrated Circuit. 20

**AVX** Advanced Vector Extensions. 23, 40, 41

**CAGR** Compound Annual Growth Rate. 9

**CISC** Complex Instruction Set Computer. 23

**CMMI** Capability Maturity Model Integration. 29

**CMOS** Complementary Metal Oxide Semiconductor. 17

**CoWoS** Chip-on-Wafer-on-Substrate. 27

**CPU** Central Processing Unit. I, IX–XI, 4, 5, 8, 15, 17, 20–26, 29–35, 37, 38, 42, 50, 51, 53, 58–64, 69, 70, 73–76, 80, 86, 88, 90–95, 97, 98

**dGPU** Discrete Graphics Processing Unit. 51, 70, 73, 86, 87, 95, 97, 98





**DPU** Data Processing Unit. 37

**DRAM** Dynamic Random Access Memory. 25

**DSL** Domain Specific Language. 59

**DSM** Digital Surface Model. 53

**DSP** Digital Signal Processor. 19, 20, 34

**DTM** Digital Terrain Model. 3, 44, 45, 50–53

**DTW** Dynamic Time Warping. i, 3, 46, 138

**EB** Exabytes. 9

**ECG** Electrocardiogram. 44

**ED** Euclidean Distance. 46

**EEG** Electroencephalography. i, 1–6, 8, 44, 46, 47, 141

**EMG** Electromyography. 47

**EU** Execution Unit. 22, 24, 25, 27

**FLOPS** Floating Point Operations Per Second. ix, 17–19, 24

**FPGA** Field-Programmable Gate Array. ii, 19, 21, 30–32, 34, 35, 46

**GPC** Graphic Processing Cluster. 27

**GPGPU** General-Purpose Graphics Processing Units. 17, 59

**GPS** Global Positioning System. 45, 49

**GPU** Graphics Processing Unit. i, ii, ix, 5, 8, 17–22, 24–27, 30–38, 46, 50–53, 58–62, 69–71, 73–75, 86, 87, 89–93, 97, 98

**HPC** High Performance Computing. 2, 11, 29, 59, 142

**HyOF** Hybrid Overlap Filter. 3, 53

**IaaS** Infrastructure as a Service. 11

**iGPU** Integrated Graphics Processing Unit. 21, 25–27, 35, 70, 73, 86, 94, 97, 98



**ILP** Instruction Level Parallelism. 15, 23, 24

**IMT** Interleaved Multi-Threading. 25

**IMU** Inertial Measurement Unit. 49

**IoE** Internet of Everything. 12–14, 19, 20

**IoT** Internet of Things. 10, 96

**IRDS** International Roadmap for Devices and Systems. 15, 16

**ISPRS** International Society of Photogrammetry and Remote Sensing. 52

**IT** Information Technology. 10–12, 29

**JIT** Just-in-Time. 34, 35

**LCSS** Longest Common Subsequence. 46

**LEO** Low Earth Orbit. 11

**LiDAR** Light Detection and Ranging. i, ii, xi, 1–6, 8, 35, 42, 46, 47, 49–53, 57, 59–62, 65, 75, 76, 92, 97, 98

**LLC** Last Level Cache. 22, 26

**LLP** Local Lowest Point. ix, 54–56, 59, 67, 75

**LPU** Language Processing Unit. 19

**M2M** Machine-to-Machine. 12

**MCU** Microcontroller Unit. ii

**MEG** Magnetoencephalography. 47

**ML** Machine Learning. 17, 21, 23, 33, 44

**MtM** More than Moore. 15–18, 28

**NPU** Neural Processing Unit. 19

**OpEx** Operating Expenses. 11

**OWM** Overlap Window Method. i, ix–xi, 3, 50–57, 59, 60, 62–64, 66–69, 73–94, 97, 137

**PaaS** Platform as a Service. 11

**PaFESD** Patterns augmented by Features Epileptic Seizure Detection. i, ii, 3, 138

**PB** Processing Block. 27, 37

**PC** Personal Computer. 17, 24

**PCIe** Peripheral Component Interconnect Express. 21, 23, 25

**RISC** Reduced Instruction Set Computer. 20, 23

**SaaS** Software as a Service. 11

**SGF** Semi-Global Filtering. 52, 92

**SIMD** Single Instruction, Multiple Data. 23, 25, 40, 41, 46

**SiP** System in Package. 16, 17, 20, 27, 28

**SM** Streaming Multiprocessor. 26, 27, 37, 38

**SMT** Simultaneous Multi-Threading. 25

**SNN** Spiking Neural Network. 13

**SoC** System-on-Chip. i, 3, 16, 20–24, 26–29, 37

**SPD** Symmetric Positive Definite. 47

**STL** Standard Template Library. 31, 38, 39

**SW** Sliding Window. 54, 55, 66, 67

**TDP** Thermal Design Power. 7, 18, 22, 25, 26, 61, 95, 97

**TPU** Tensor Processing Unit. 19

**UCIe** Universal Chiplet Interconnect Express. 27

**USM** Unified Shared Memory. 31

**VNNI** Vector Neural Network Instructions. 23

**VR** Virtual Reality. 29



**XaaS** Everything as a Service. 10, 11

**XR** Extended Reality. 12

**ZB** Zettabytes. 9

# 1 Introduction

Data ignites our world.As we navigate through the era of big data, the art and science of massive data analytics emerge as a pivotal force, transforming extensive datasets into actionable insights across diverse formats. In particular, massive sensor-based data analytics encompasses the processing and analysis of extensive datasets that manifest in various forms, including structured time series, unstructured text, and complex multimedia information. These datasets are integral to a wide array of scientific and engineering disciplines—ranging from environmental monitoring and biomedical engineering to smart manufacturing, where they provide valuable insights into underlying dynamics and structure of phenomena. This requires a balanced approach that takes into account the volume, variety, and the inherent complexity, dynamism, and non-linearity of the data they contain. To effectively tackle the challenges presented by massive datasets, it is imperative to develop and apply sophisticated data representations and methods for effectively capturing, modeling, and interpreting relevant information, ensuring both the accuracy and the efficiency of the analysis.

The main goal of this thesis is to develop and apply novel methods and tools for information extraction and analysis from large-scale data collections, with a particular focus on two specific sensor-based applications: Light Detection and Ranging (LiDAR) technology for collecting accurate Earth's landscape data and epileptic seizure detection using Electroencephalography (EEG). In the early stages of this research, we wondered what kind of applications could benefit from advanced algorithmic approaches and architectural optimizations. We discovered that while fields like image processing, Photogrammetry, ECG analysis, or speech recognition have seen significant advancements in processing efficiency, other crucial areas lagged behind [53, 98]. Notably, LiDAR technology and EEG-





based epileptic seizure detection stood out as applications where the algorithmic approaches focused on accelerating their processing, while maintaining power efficiency and real-time constraints, were far behind other fields. Interestingly, we found that these two techniques (LiDAR and EEG) offer superior accuracy compared to related methods in their respective domains. LiDAR provides unparalleled precision in 3D mapping and object detection [98], while EEG offers the highest temporal resolution and the most direct and accurate method for monitoring brain activity [53]. However, their widespread adoption has been limited by factors such as high costs, processing times, preprocessing requirements, and the complexity of the environments in which they operate. We recognized an opportunity to address these gaps and potentially contribute to make these technologies more accesible. These applications, while distinct in their nature, share a common goal of enhancing the quality and reliability of information through advanced data analytics. Given the widespread availability of power-efficient heterogeneous computing platforms today, a key question arises: How can we effectively leverage these systems to accelerate these applications? This exploration is particularly challenging due to the inherent power consumption and real-time constraints of sensor-based applications. It necessitates innovative approaches that balance between analytical depth, computational efficiency, and power management. Consequently, the primary focus of this work is to address the following research question:

> *"How can High Performance Computing (HPC) (high performance programming and high performance architectures) be leveraged to optimize massive sensor-based data processing across different applications?"*

As previously mentioned, this work is divided into two main parts, each focusing on a unique application of sensor-based data analytics. The initial segment delves into the LiDAR active sensing technique, a pivotal method for capturing precise Earth's landscape data. Compared to traditional acquisition methods like aerial imagery, LiDAR offers superior accuracy, with the potential to achieve point cloud densities of over 100 points per square meter, significantly enhancing terrain representation and object detection. This high-resolution data collection, however, presents substantial challenges, particularly in data volume and processing demands. For instance, a single LiDAR survey can generate several gigabytes of data, necessitating advanced algorithm optimization and parallelization to efficiently process such massive datasets. Currently, our methodology for processing LiDAR data is implemented in an offline manner. This approach allows for meticulous and comprehensive analysis of the data, ensuring accuracy and reliability. However, we acknowledge the increasing demand and potential

for real-time data processing in various applications. Accordingly, our proposal is designed with a forward-looking perspective, with the aim of evolving into a real-time processing methodology in future work.

Central to our discussion is the Overlap Window Method (OWM), a foundational technique for ground point extraction from Airborne Laser Scanning (ALS) point clouds. This method is not only a critical component of our data processing pipeline but also serves as the initial step in a more comprehensive algorithm known as the Hybrid Overlap Filter (HyOF). The significance of OWM extends beyond mere data reduction; it embodies a strategic application of information theory by filtering out non-essential points, thereby simplifying the dataset while preserving crucial information for Digital Terrain Model (DTM) creation. For our implementations of the OWM we explore various programming models that represent the state-of-the-art in heterogeneous computing, including SYCL, SYCL+CUDA, and pure CUDA, finding promising results both in terms of performance and portability.

Transitioning to the second part of the thesis, we address epilepsy, a disorder affecting over 50 million people worldwide. EEG data, crucial for epilepsy diagnosis and monitoring, present unique challenges such as variability, non-stationarity, and high dimensionality. These challenges affect the accuracy and reliability of seizure detection, reinforcing the need for advanced analysis techniques. In this work, we introduce Patterns augmented by Features Epileptic Seizure Detection (PaFESD), a novel approach that combines Dynamic Time Warping (DTW) with feature-based signal analysis for accurate seizure identification in EEG data. This method exemplifies the balance between theoretical rigor and experimental pragmatism, highlighting the importance of architectural considerations, in particular, for the implementation of epileptic seizure detection on a wearables, based on a low-power heterogeneous System-on-Chip (SoC).

This transition from general principles of massive sensor-based data analysis to the specific applications of LiDAR and EEG demonstrates the aim of this thesis. While the data types and methods of analysis may differ, the underlying objective remains consistent: to improve the quality and reliability of information through the application of advanced analytical strategies, and their porting to heterogeneous computing architectures using state-of-the-art heterogeneous programming models. The practical implications of this research are significant, offering potential advancements in LiDAR data processing efficiency and the accuracy of epilepsy diagnosis, thereby impacting real-world applications and enhancing the understanding of complex datasets.

## 1.1. Motivation

In an era characterized by a significant increase in data production from sensor-based applications, the ability to efficiently process, analyze, and extract valuable insights from large datasets and real-time streams is crucial for innovation and for achieving competitive advantage. This phenomenon of "data deluge" presents both challenges and opportunities across diverse sectors. The development of advanced computational methodologies, algorithms, programming models and architectures capable of handling the increasing scale and complexity of data is central to navigating this landscape. This research addresses these pressing needs through a dual focus on optimizing LiDAR data processing and enhancing epileptic seizure detection through EEG analysis. The motivation for this research is driven by the following key factors:

- **The Challenge of Scale and Complexity:** Traditional data processing tools and methods, including LiDAR data processing, are challenged by today's data volumes. To manage massive datasets in real-time applications, from autonomous vehicles to environmental monitoring, the use of energy efficient heterogeneous architectures that feature Central Processing Unit (CPU) and diverse accelerator devices, as well as the efficient integration of specific device optimizations through portable and productive programming models, are critical.
- **The Opportunity for Innovation:** The massive volume of data provides unprecedented opportunities for breakthroughs in various fields. This research aims to capitalize on these opportunities by advancing LiDAR data processing techniques, and improving epileptic seizure detection through innovative EEG analysis models, which can lead to significant advances in edge computing and real-time data processing.
- **Interdisciplinary Implications:** The impact of advanced data analytics goes beyond individual disciplines, influencing scientific research, business models, and the development of new laws. This research takes an interdisciplinary approach, combining insights from computer science, computer architecture, geospatial science, and neuroscience, to demonstrate the comprehensive strategy needed to fully leverage the opportunities presented by massive sensor-based data analytics.
- **The Role of Technological Innovation:** The rapid evolution of computing architectures and software technologies, including heterogeneous programming models, is the foundation for the advancements in massive sensor-based data analytics. This research utilizes these innovations, specifically parallel computing techniques, up-to-date programming models, and mem-

ory optimization strategies adapted to the specific capabilities of the specialized computing devices, to effectively address the challenges presented by large-scale data environments.

## 1.2. Objectives

Within the context of addressing the challenges and opportunities presented by the vast amounts of sensor-based data generated in the selected applications of interest, this research aims to achieve the following specific objectives:

1. **Advance LiDAR Data Processing Techniques:** By developing state-of-the-art optimization strategies for CPU and Graphics Processing Unit (GPU) devices, the goal is to improve the efficiency and accuracy of LiDAR data analysis. Scalable algorithms will be created that leverage novel approaches in data structure representations, parallel computing and data traversal operations to overcome the computational challenges posed by large LiDAR datasets. Advancements like these enable real-time processing capabilities in fields such as autonomous navigation and environmental surveillance.
2. **Advance Epileptic Seizure Detection through EEG Analysis:** The purpose is to enhance the methodologies for detecting epileptic seizures by analyzing EEG data through advanced pattern recognition and feature extraction techniques. Our focus is on developing metrics and patterns that can inform the design of future algorithms capable of real-time detection. The algorithm's outcomes are crucial in establishing a foundation for more dependable, efficient, and scalable seizure detection methods in the future. The insights gained from this research will have significant implications for the development of monitoring systems used in wearable devices and clinical settings.
3. **Interdisciplinary Collaboration:** Integrating methodologies from different scientific fields, such as computer science, computer architecture, geospatial science, and neuroscience, demonstrates the benefits of tackling complex challenges in sensor-based data analytics. By utilizing principles and insights from these diverse disciplines, innovative approaches are developed that go beyond traditional boundaries. Collaborative efforts enrich understanding and set the stage for comprehensive and effective solutions. This interdisciplinary approach demonstrates the benefits of diverse scientific perspectives converging.
4. **Leverage Technological Innovations:** This goal addresses the scal-

ability and efficiency challenges inherent in processing large-scale sensor datasets. We explore cutting-edge approaches in parallel processing, algorithmic design, and novel programming models to optimize data handling and analysis. Our research focuses on evaluating how state-of-the-art computing platforms can enhance sensor-based data processing and analysis across various domains. Specific applications include LiDAR data processing, management of irregular data structures, pattern recognition, and feature extraction.

## 1.3. Methodology

The methodology employed in this thesis is characterized by a multifaceted approach that combines theoretical analysis, algorithmic development, and empirical evaluation. This approach is designed to address the complex challenges presented by massive sensor-based data analytics, the inherent constraints of the real-time processing applications, and the need for power efficiency, particularly in the domains of LiDAR data processing and EEG-based epileptic seizure detection. Our research methodology can be summarized in the following key steps:

1. **Literature Review and Problem Formulation:** We begin with a comprehensive review of existing literature in the fields of LiDAR data processing, EEG analysis, and high-performance computing. This review helps identify current challenges, state-of-the-art techniques, and potential areas for improvement. Based on this analysis, we formulate specific research questions and objectives.
2. **Algorithm Design and Development:** Building on the insights gained from the literature review, we design novel algorithms and improve existing ones. For LiDAR data processing, we focus on optimizing ground point extraction techniques. In EEG analysis, we develop new approaches for epileptic seizure detection. These algorithms are designed with a focus on efficiency, accuracy, and scalability.
3. **Implementation and Optimization:** We implement our algorithms, and design comprehensive experiments, using various programming models and languages, including Python, C++, SYCL, CUDA, and oneTBB. This stage involves careful consideration of hardware architectures and programming paradigms to maximize performance and energy efficiency. We employ techniques such as parallelization, memory optimization, and architecture-specific tuning.
4. **Performance Evaluation:** We conduct extensive experiments to assess



the performance of our algorithms. This includes measuring execution time, accuracy, and throughput. We estimate power efficiency by extrapolating from the Thermal Design Power (TDP) of the devices, rather than directly measuring energy consumption. We compare our results against baseline methods and state-of-the-art techniques to quantify improvements.

5. **Analysis, Interpretation, and Iterative Refinement:** The experimental results are analyzed and validated to ensure the robustness of our findings, and to draw meaningful conclusions, interpreting the findings in the context of our research objectives and the broader field of massive sensor-based data analytics. Based on this analysis, we iteratively refine our algorithms and implementations. This step is crucial as it's where we integrate knowledge and principles from other disciplines to achieve significant performance gains. This interdisciplinary synthesis may involve revisiting earlier steps in the methodology to address identified limitations, explore novel optimization opportunities, or incorporate insights from other fields.

Throughout this process, we maintain a strong focus on the practical applicability of our research. Our methodology also emphasizes the interdisciplinary nature of this research. We integrate insights from computer science, information theory, and domain-specific knowledge in geospatial science and neurology. It is important to note that our primary motivation in this research is the pursuit of knowledge and understanding, rather than the validation of preconceived notions. We approach our work with an open mind, recognizing that scientific progress often involves challenging established methodologies and embracing unexpected outcomes.

The history of scientific discovery teaches us that even foundational theories can take centuries to be rigorously proven, as exemplified by the Central Limit Theorem [32]. From de Moivre's initial work in 1733 to Kolmogorov's formulation of necessary and sufficient conditions for the Law of Large Numbers in 1928, this theorem's development spanned nearly two centuries. This historical perspective reminds us that the application of theoretical concepts can precede their formal proof, and that persistence in the face of complex challenges is a hallmark of scientific progress.

Furthermore, we acknowledge that rigid adherence to established methodological rules can sometimes hinder scientific advancement. As philosopher Paul Feyerabend argued [31], many significant scientific breakthroughs occurred because researchers were willing to challenge or unwittingly break conventional methodological rules. This insight encourages us to maintain a degree of flexibility in our approach, allowing for creative problem-solving and innovative thinking.

Lastly, we recognize the limitations of reductionist approaches in analyzing complex systems. As complexity scientist David Krakauer noted [78], "reductionism in the units of analysis not only fails to explain complexity; it fails to detect it". This understanding guides our holistic approach to massive sensor-based data analytics, where we strive to consider the interconnected nature of the systems we study and the multifaceted challenges they present.

## 1.4. Document structure

The structure of this thesis is organized as follows. Chapter 2 provides a comprehensive overview of the foundational concepts and technologies relevant to this research. It explores the evolution and current state of data analytics, with a particular focus on the exponential growth of sensor-based data and its implications for data-driven science and technology. This chapter sets the stage by contextualizing the challenges and opportunities that massive sensor-based analytics presents across the domains of computer architectures, heterogeneous systems, and algorithmic approaches. Chapter 3 delves into the specific methodologies and optimizations for LiDAR data processing, emphasizing the porting to CPU and GPU architectures, and the device-specific optimizations to enhance the computational and energy efficiency as well the accuracy of data analysis. The discussion includes the analysis of the data structures that better exploit the device architectural features, the design of methodologies that minimize memory accesses and optimize memory bandwidth usage, as well as the development of scalable algorithms using state-of-the-art heterogeneous computing frameworks. Chapter 4 shifts focus to EEG-based epileptic seizure detection, outlining the core methodologies used in this area. It evaluates current approaches and proposes improvements, emphasizing the combination of pattern recognition and feature extraction techniques to advance seizure detection methodologies. We provide some insights about the complexity of the evaluated approaches as well as the real-time constraints of our seizure detection algorithm when ported to an ultra-low-power wearable devices. In Chapter 5, conclusions are drawn from the research, summarizing the contributions made in the fields of LiDAR data processing and epileptic seizure detection through EEG analysis. This chapter reflects on the impact of these advancements on future research directions and practical applications. Finally, Chapter 6 outlines future work, presenting a vision for extending the research findings. It proposes potential avenues for advancing the interdisciplinary approaches developed in this thesis and explores how these methodologies could be applied to address new challenges in massive sensor-based data analytics and real-time data processing.



# 2 Background

## 2.1. The Data Deluge

Recent studies and market analyses [4, 29, 61] have highlighted the significant growth of the global data-analytics market. As of 2023, the market's valuation was approximately USD 310 billion, with projections suggesting a Compound Annual Growth Rate (CAGR) of 14.6% from 2024 to 2032. This projected growth anticipates a market size of above USD 1 trillion by the conclusion of 2032. Such expansion not only underscores the escalating reliance on data analytics across various sectors but also the integration of advanced technologies and methodologies pivotal to the field's future. Forefront innovations (including real-time stream processing, AI-enhanced trust, risk, and security management, continuous threat exposure management, and the engineering of sustainable technology platforms) are driving this evolution. Their influence is substantial, transforming practices in domains such as marketing, transportation, governance, business operations, healthcare, and cybersecurity.

Parallel to the expansion of the data-analytics market is the exponential growth in data generation. In 2023, global data production was estimated at approximately 120 Zettabytes (ZB), with forecasts predicting an increase of over 50% by 2025, reaching 180 ZB [52, 65]. This growth translates to an average daily output of around 490 Exabytes (EB), with a CAGR of 20% [64]. Specifically, in mobile data, future projections [119] suggest that monthly global data traffic will reach 607 EB by 2025, with an anticipated escalation to 5 ZB by 2030. This reflects an annual growth rate of 55% between 2023 and 2030, un-

derscoring the burgeoning challenges and opportunities in managing, analyzing, and utilizing data on such a massive scale. Considering the significant environmental impact highlighted by the increasing energy demands of data centers and the digital sector's contribution to global greenhouse emissions, it is crucial to develop and adopt low-power computer architectures and power-efficient processing methodologies [68]. These strategies can effectively address the issue at hand with confidence and expertise. These advancements are crucial for reducing the environmental impact of large-scale data analytics, aligning technological growth with sustainability goals, and promoting eco-friendly data management practices to achieve progress towards net-zero emissions.

The unprecedented surge in data generation witnessed in 2023 is attributable to various sources, including advancements in Artificial Intelligence (AI), the proliferation of the Internet of Things (IoT), widespread usage of social media, developments in cloud computing technologies, enhanced satellite imagery capabilities, and the deployment of biometric sensors. This assortment of data sources has led to the emergence of both structured and unstructured datasets of unparalleled size and complexity, necessitating advanced methods for efficient storage, processing, and analysis. Notably, AI emerges as the primary driver of this massive data production, introducing unique challenges and opportunities for data management strategies. In alignment with this trend, a report from the International Data Corporation (IDC) [64] predicts a significant shift in Information Technology (IT) budgets towards AI technologies, indicating a sector-wide transformative change. By 2025, it is anticipated [63] that organizations within the Global 2000 (G2000) (the world's largest and most influential companies) will dedicate over 40% of their core IT budgets to AI-focused initiatives. This strategic investment is expected to spur double-digit percentage growth in product and process innovation rates, highlighting the profound influence of AI on operational efficiencies and competitive dynamics in the global market. Complementing this shift, a proposed cloud-edge computing framework [85] aims to efficiently handle IoT-generated high-volume data streams by distributing processing tasks. This approach not only aligns with the IT industry's move towards AI-driven efficiency but also underscores the balance between leveraging edge computing for immediacy and cloud resources for comprehensive analysis. This represents a significant advancement in computer architecture and algorithms, crucial for navigating the complexities of large-scale data analytics, and highlights the IT industry's adaptive strategies in optimizing data processing infrastructures for the digital age.

At the heart of the IT industry's ongoing digital transformation is the emerging paradigm of Everything as a Service (XaaS). This model is poised to revo-

lutionize business operations, enabling enterprises to focus on their core competencies while utilizing externally managed services for their IT requirements [62]. The pivot towards XaaS calls for a shift away from traditional computing frameworks, which are becoming increasingly insufficient for meeting the growing demands for data processing and computational power. Consequently, there is an urgent need for the implementation of innovative, data-centric architectural designs. These designs are tailor-made to support the burgeoning suite of data-intensive applications and services, ensuring scalability, efficiency, and agility in data management and processing. Future projections [63, 66] suggest that by 2027, the predilection for Operating Expenses (OpEx)-based computational services (characterized by subscription models such as cloud computing, Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS)) will drive significant advancements in quantum computing, AI, and High Performance Computing (HPC) within the public cloud service domain. This transition highlights a strategic movement towards more adaptable, scalable, and cost-effective computing paradigms, positioning quantum computing, AI, and HPC as key elements in the forthcoming generation of cloud-based services. This progression is anticipated to not only transform the IT service delivery landscape but also to catalyze unparalleled levels of innovation and efficiency in computational methodologies.

As demonstrated thus far, the current era is distinguished by an unparalleled influx of data, characterized by significant volume, variety, and velocity, originating from diverse sources. This deluge has profoundly altered societal structures, attracting attention from both specialists and the wider public. To adeptly navigate this data-saturated environment, cutting-edge data processing techniques are essential. Among these, the *wireless first* strategy emerges as a critical methodology [67]. It capitalizes on the exponential rise in wireless devices and applications, such as smartphones, tablets, and IoT devices, where Low Earth Orbit (LEO) satellite connectivity plays a pivotal role, providing a cost-effective, low-latency alternative for expanding wireless access. Projections indicate [91, 118] an increase to 207 billion devices globally, with related expenditures expected to surpass USD 1.3 trillion by 2024. This strategy emphasizes a shift towards prioritizing wireless connectivity in data acquisition and processing, enabling real-time insights and decision-making across numerous sectors. This approach is particularly relevant in the realm of massive sensor-based data analytics, where the incorporation of wireless technologies enhances the efficiency and scalability of data processing systems. The swift transformation of IT firms is compelled by the emergence of groundbreaking technologies like 6G, WiFi 7, and Augmented Reality (AR), highlighting the imperative to transition towards

a data-centric paradigm.

Amidst the data-centric innovation era, prestigious grants and fellowships globally act as catalysts, supporting research that advances power-efficient solutions across digital platforms, systems, and architectures. This financial and academic support drives advancements to address the challenges of massive data influx and swift IT industry adaptation. It is worth noting that governments worldwide are significantly funding these technologies. Through the European Chips Act, the European Union will mobilize more than 43 billion euros of public and private investments in next-generation technologies to design tools and pilot lines for the prototyping, testing and experimentation of cutting-edge chips, as well as energy-efficient and trusted chips to guarantee quality and security for critical applications [28], while the US has proposed the "CHIPS and Science Act" to propel semiconductor, AI, and quantum computing research with a budget of nearly USD 39 billion [33, 124]. Similarly, China has committed to investing over USD 40 billion to enhance its semiconductor capabilities [120]. These efforts reflect a global push towards sustaining novel efficient computing platforms and architectures tailored to the needs of emerging applications.

However, the data-centric shift presents considerable challenges, especially for IT infrastructure [27], which is now required to support a surge in data-intensive applications and services. These include AI, the Internet of Everything (IoE), extensive Extended Reality (XR) implementations, digital twins, and ubiquitous intelligence. Acknowledging the extensive discussion on data proliferation, the real challenge transcends adapting traditional models, highlighting a comprehensive need across platforms, systems, and architectures for innovative, power-efficient solutions.

## 2.2. Opportunities for Data-Driven Science

In the evolving landscape of data-driven science, applications of the IoE are recognized as instrumental in driving digital transformation across various sectors. The introduction of IoE wearables, such as smartwatches, glasses, and clothing, represents a significant advancement in edge computing, offering real-time insights into our environments, health, and activities. With the number of licensed cellular IoT connections expected to double reaching 5.3 billion by the end of this decade [41], the extent of digital integration into everyday life is anticipated to increase substantially. Expanding beyond conventional Machine-to-Machine (M2M) communications, the IoE paradigm envisions the integration of people, robots, and machines into a unified system via the Internet, thereby

generating an enormous volume of data. Addressing this deluge of data necessitates innovative strategies that leverage computing resources situated closer to the sources of data and end-users. Nevertheless, this pivot towards *near/in-sensor* computing presents several challenges, including the imperative for real-time data processing, enhanced energy efficiency, and the development of new architectural systems capable of accommodating these sophisticated functionalities, an issue that we target in this thesis.

Energy efficiency stands as a critical factor in our domain of interest: massive sensor-based data processing, and it affects both hardware devices as communication protocols. Its significant influence on system durability, especially in IoE contexts where replacing batteries may be impractical, highlights energy conservation as a major challenge within both hardware and communication layers. The emergence of entirely autonomous wearables powered by renewable energy sources emphasizes the necessity of processing data close to the sensor. In this regard, edge computing becomes crucial, facilitating effective data management at the network's edge. Advances in hardware design and computational models offer promising avenues for substantially reducing energy consumption, in particular the emergence of heterogeneous architectures that contain multiple processing units, which are becoming the systems of choice in several edge computing applications, and specifically in our applications . Other examples of novel energy efficient architectures are the implementation of Spiking Neural Network (SNN), which require approximately $\sim$1nW per operation, enables efficient in-memory processing for applications like heartbeat arrhythmia detection [19]. Further developments in transistor technology, especially the 3D integration of 2D nanosheets and leveraging thermodynamic entropy in the logic scaling assembly processes, signal the advancement of *near/in-sensor* computing architectures [70, 130]. These technological breakthroughs are instrumental in developing energy-efficient edge computing solutions, heralding substantial energy savings and improved system performance.

The vast amount of data and the need for faster processing have resulted in many new challenges for the chip industry [23]. These challenges are not immediately apparent or easily solvable. As the chip industry continues to integrate diverse features in novel ways to cater to specific markets, power-related issues such as architecture, place-and-route, signal integrity, heat, reliability, manufacturability, and aging will become increasingly prevalent and interconnected. Therefore, the entire industry will need to adapt and develop strategies to manage or circumvent these power-related effects. Even for lower-volume chip makers, ignoring power considerations will become increasingly rare, and they too will need to address these issues.

## 2.3. Hardware evolution

As we dissect the intricacies of energy-efficient data processing strategies, it becomes imperative to examine the underlying hardware facilitating these advancements. *Microprocessors*, the foundation of *near/in-sensor computing* architectures, empower the processing capabilities essential for handling the vast volume of data produced by IoE devices. The evolution of microprocessors over the past five decades has been pivotal, showcasing significant enhancements in performance, energy efficiency, and integration levels, as illustrated in Figure 2.1. The seminal observation made by Gordon E. Moore in 1965, which underscored that "the complexity [of integrated circuits] for minimum component costs doubles approximately every two years", laid the groundwork for the evolutionary trajectory of microprocessors [95]. A decade later, Moore refined this projection to a biennial doubling, a principle that has been immortalized as "Moore's Law" [96]. Despite occasional deviations, Moore's Law has acted as a reliable macro trend, guiding the development of cutting-edge semiconductor products for over fifty years. The exponential enhancement in microprocessor performance noted by Moore finds reflection in the developments illustrated in Figure 2.1. Yet, the onset of the new millennium introduced a formidable challenge known as the "power wall". This obstacle arose as the relentless miniaturization of digital components, both logic and memory storage, aimed at increasing density and reducing cost per function, led to a critical juncture in power consumption. This trend, together with heightened operating frequencies, not only augmented performance but also significantly increased power demands. The resultant power density reached a threshold where the heat generated by the chips outstripped their cooling capabilities, leading to a thermal bottleneck. This limitation has notably slowed the pace of frequency improvements, culminating in a stabilization around the 3 GHz mark, a phenomenon clearly evidenced in Figure 2.1 (green dots).

In 1974, another seminal contribution to the field was made by engineer Robert H. Dennard, who introduced the concept of "Dennard scaling" [21]. This principle asserted that with the miniaturization of transistors, power density across a chip would remain constant, thereby allowing power consumption to scale down proportionally with the chip's area. However, this paradigm faced a significant obstacle in the 2000s with the advent of the "power wall". The continuous trend towards miniaturization made it increasingly difficult to lower the voltage and current at which transistors operate without affecting their reliability. The best illustration of this challenge is the recent news form Cerebras Systems, which announced the development of the world's largest chip, the Cerebras CS-3, measuring 46,225 $mm^2$ and containing 4 trillion transistors using the

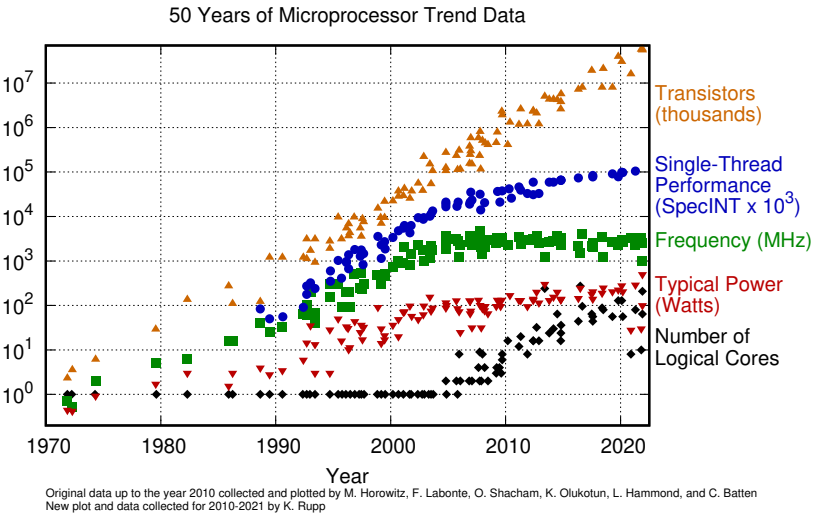



Figure 2.1: 50 years of CPU processors evolution [105].

5 nm TSMC's technology. This chip, designed for AI workloads, is a testament to the industry's ongoing efforts to overcome these physical limitations. While Moore's Law and its derivatives have been interpreted and adapted in various ways over the decades, often with remarkable foresight, a recurring limitation of many of these predictions has been an underestimation of technological constraints. This oversight resulted in forecasts that, despite their innovation, were ultimately deemed unsustainable as they overlooked the inherent limitations of existing technologies.

In 2005, the International Roadmap for Devices and Systems (IRDS) [56] unveiled the groundbreaking concept of More than Moore (MtM), stemming from the acknowledgment that various essential architectural needs (such as power consumption, wireless communication capabilities, integration of passive components, along with sensing, actuating, and biological functions) do not conform to the scaling principles outlined by Moore's original observation. The MtM paradigm goes beyond the traditional focus on transistor miniaturization, advocating for the incorporation of a wide range of functionalities within devices. This strategy aims to augment value through the delivery of multifunctional, heterogeneous system solutions, incorporating advanced concepts like memory hierarchy, Instruction Level Parallelism (ILP), and a broad array of multidisciplinary tech-

nologies spanning the entire spectrum of innovation. Known as "Equivalent Scaling", this approach acts as a complement to "Geometrical Scaling" (aligned with Moore's Law or "More Moore", as delineated in the IRDS report [56]) driving the progression of System-on-Chip (SoC) and System in Package (SiP) technologies. By moving beyond the confines of conventional miniaturization, MtM has positioned itself as a foundational element in the continuous evolution of complex, integrated systems. This shift signifies a substantial transformation in the semiconductor industry's scaling and innovation methodologies.

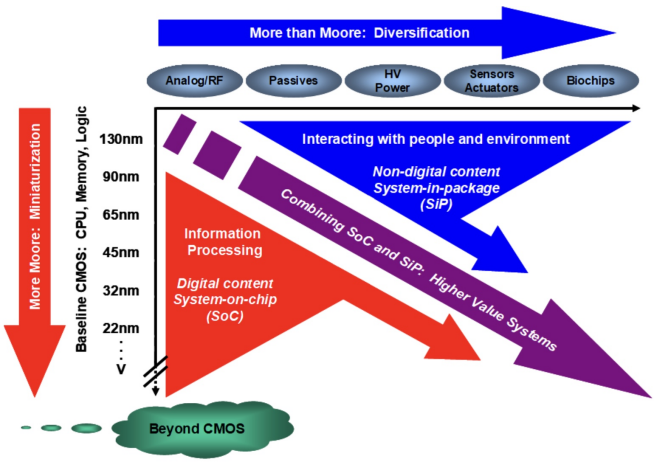



Figure 2.2: Moore's law and more [56].

The concept of MtM, as depicted in Figure 2.2, represents a paradigm shift in the understanding of technological progress. This conceptual framework delineates the synergistic relationship between "More Moore" and MtM, proposing not a dichotomy but a complementary integration of these technological paths to forge innovative high-value systems. The figure illustrate a bifurcated evolutionary trajectory: the vertical axis symbolizes the continuous miniaturization and enhancement of efficiency within digital devices (a phenomenon encapsulated by Moore's law). In contrast, the horizontal axis captures the essence of the MtM approach, emphasizing technology diversification through the amalgamation of heterogeneous components, including smart sensors, high-efficient power devices, energy harvesting, and advanced packaging technologies such as wearable, flexible, and printed electronics. This holistic integration is pivotal in crafting systems that possess an augmented capacity to interface with their environment and users, extending the capabilities of such systems through SiP solutions that seamlessly converge with SoC designs. MtM thus transcends the confines of conventional

Complementary Metal Oxide Semiconductor (CMOS) scaling and ventures into a broader area of innovation. It fosters the creation of systems that amalgamate digital progress with multifunctional attributes, thereby delivering superior value. Uniquely, MtM advocates for a comprehensive and interdisciplinary approach to innovation, spanning the entire innovation chain [56]. This positions MtM not only as a quantifiable metric, but also as a philosophical framework to guide technological evolution, advocating for a holistic view of progress that is technology agnostic, yet capable of enabling cross-cutting comparisons across a spectrum of systems, from Central Processing Unit (CPU) and Graphics Processing Unit (GPU) to SiPs, Personal Computer (PC), wearables, and even supercomputers. In response to this imperative, Wu-chun Feng introduced in 2003 a metric that realigns the focus from mere performance to a balanced consideration of efficiency, reliability, and availability [30]. By advocating for FLOPS/W (Floating Point Operations per Second per Watt) as a critical metric, Feng emphasizes the critical importance of *hardware-software* co-design. This approach not only advocates optimizing energy efficiency, but also ensures that performance is not compromised in the pursuit if sustainability and broader applicability. This metric embodies the principles of the MtM philosophy by prioritizing a comprehensive evaluation of system capabilities, thereby extending the conceptual framework of technological innovation beyond conventional metrics and scaling laws. Subsequent to this proposal, initiatives like "Koomey's law" emerged [133], furthering the discourse on computing energy efficiency. Koomey's law, focusing on the rate at which computational power per unit of energy doubles, serves as a complement to Feng's FLOPS/W metric by providing a quantifiable trend that underscores the exponential improvements in energy efficiency over time. These efforts collectively contribute to a nuanced understanding of technological progress, moving beyond mere performance metrics to encompass the efficiency and sustainability of computing systems.

In the pursuit of energy-efficient computing, the evolution of hardware architectures, best represented lately by the advent of GPUs focused on AI workloads as shown in Figure 2.3, has been instrumental [47]. The Figure 2.3, shows that the rate of growth of computational efficiency is not tapped out, with the energy efficiency of Machine Learning (ML) GPUs doubling every 3.0 years, compared to 2.70 years for General-Purpose Graphics Processing Units (GPGPU). This trend indicates that energy consumption is not currently a bottleneck to scaling, particularly supported for the new fabrications techniques, the development of new architectures, and because energy consumption is not yet a bottleneck to scaling. However, everything points to the fact that it may very well be a concern in the future [46].

Despite these advancements in conceptualizing and measuring technological progress, skepticism persists regarding the long-term sustainability of such trends [45], especially as they approach the physical and thermodynamic limits of computing technologies. This skepticism is rooted in the recognition that while laws like Moore's and Koomey's have historically capture the trend of technological progress, the inevitable approach to fundamental physical constraints poses significant challenges. The concern lies not just in maintaining the pace of improvement but in the broader question of how technology can continue to evolve within these finite bounds, prompting a reevaluation of what constitutes sustainable progress in the face of these limits.

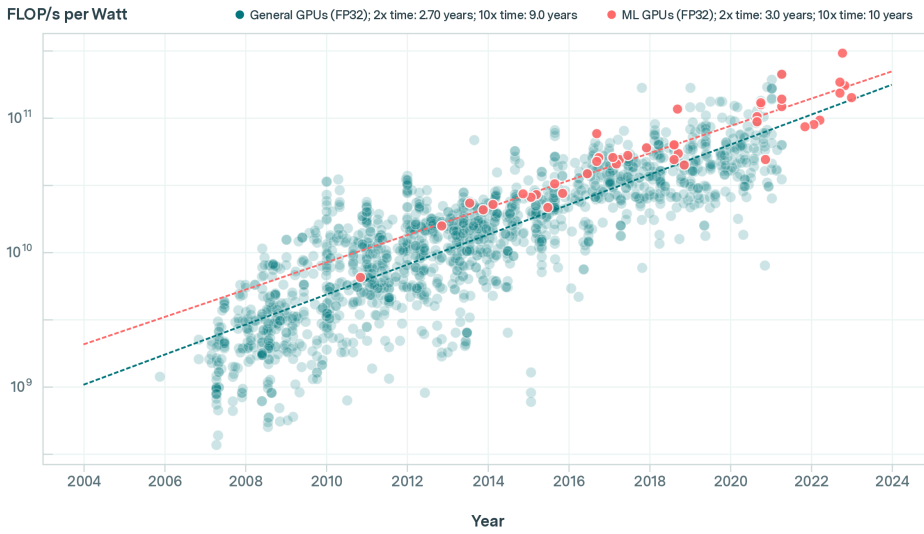



Figure 2.3: Evolution of computational efficiency in FLOPS/W for FP32 operations, where the energy component is computed from the TDP of each GPU [47].

These evolving trends, spanning from Moore's Law and MtM to the metrics of energy efficiency, collectively emphasize the principle of "Accelerating Change" [42, 132]. This idea asserts that technological advancement is on an exponential curve, indicating that we are on the cusp of witnessing continuous, rapid progress in technology, irrespective of the specific benchmarks applied to measure such advancements. The concept of exponential growth in technology predates Moore's law, with its roots tracing back to 1910 when Daniel Burnham stated, "But it is not merely in the number of facts or sorts of knowledge that progress lies: it is still more in the geometric ratio of sophistication, in the geometric widening of the sphere of knowledge, which every year is taking in a

larger percentage of people as time goes on" [123]. This reflection on the historical trajectory of technological evolution underscores an exponential pattern, highlighting the limitations of traditional metrics while elevating the significance of energy efficiency within the"Accelerating Change" paradigm. In this context, FLOPS/W emerges not merely as a critical measure but as a central, foundational benchmark. By foregrounding energy as an essential and enduring aspect inherent to all physical processes, governed by the immutable laws of thermodynamics, the prioritization of FLOPS/W as a measure underscores our dedication to harmonizing technological advancements with the core principles that govern the natural world. Consequently, the imperative extends beyond merely identifying areas of energy waste; it encompasses the development and deployment of strategies for energy conservation, ensuring that technological progression is both sustainable and aligned with fundamental ecological and physical realities.

### 2.3.1. Chiplet architectures

The evolution towards addressing massive sensor-based data processing challenges and improving energy efficiency in IoE applications and wearable devices requires more than just incremental improvements. The technological landscape is moving towards heterogeneity, increased specialization, and a focus on energy optimization. This shift has led to the development of various processing units, each designed for specific functionalities:

- **Graphics Processing Units (GPUs)**: Specialized for parallel processing, ideal for graphics rendering, simulation, data processing, and executing complex algorithms.
- **Digital Signal Processors (DSPs)**: Optimized for high-speed numerical operations, crucial in signal processing, audio and video compression, and real-time data analysis.
- **Neural Processing Units (NPUs)**: Designed to accelerate AI applications, specializing in machine learning and deep learning tasks.
- **Language Programmable Units (LPUs)**: Tailored for natural language processing tasks, focusing on analyzing and understanding human languages [1].
- **Tensor Processing Units (TPUs)**: Developed for tensor calculations, significantly speeding up machine learning workflows for deep learning computations.
- **Field-Programmable Gate Arrays (FPGAs)**: Offering programming flexibility, reconfigurable for a range of applications from digital signal processing to custom computing tasks.



- **Application-Specific Integrated Circuits (ASICs)**: Custom-designed for particular uses, delivering optimal performance and energy efficiency in specific tasks.
- **SoC** and **SiP** devices: Integrating multiple components into a single chip or package, supporting compact, high-performance computing solutions.

These technological innovations are leading the way in efficiently managing the large amounts of data associated with IoE and wearable technologies. They are designed to meet the complex computational requirements of modern workloads and applications, with a focus on energy conservation. These innovations represent a departure from one-size-fits-all approaches and usher in an era of customized computational architectures.

For IoE applications and wearable devices, solutions that encapsulate both efficiency and performance are paramount. SoC designs and Reduced Instruction Set Computer (RISC) architectures[1] emerge as the most fitting solutions in this context. Notably, SoC and RISC architectures, exemplified by ARM and RISC-V, are particularly suited for IoE wearables. They offer an optimal balance of high performance, low power consumption, and minimal die area, aligning with the essential requirements for wearable technology as outlined in Section 2.2. These architectures distinguish themselves in the IoE wearable sector by integrating diverse cores, such as those in the DynamIQ big.LITTLE[2] configuration, and integrating multiple functional units including GPUs, DSPs, and NPUs within a single chip. This multifaceted integration effectively reduces energy dissipation and minimizes heat generation. Moreover, the adoption of streamlined and efficient instruction sets in SoC and RISC architectures enables faster processing and lower energy consumption, thus providing a sophisticated solution for addressing the intricate requirements of IoE and wearable devices [90].

The vanguard SoC technology, especially designed for mobile applications, marks a critical shift in the industry towards more specialized and heterogeneous computing frameworks. Prime examples of this evolution are represented by the Apple M3 SoC family [7] and the Qualcomm Snapdragon W5+[102], both emblematic of the trend towards integrating diverse computing elements within a single chip. These SoCs employ the big.LITTLE architecture, a dedicated GPU, and various co-processors, fabricated using cutting-edge 3-nm and 4-nm processes, respectively. The Apple M3 SoCs stand out for their advanced GPU

[1] RISC represents a CPU design philosophy that emphasizes the advantage of using a smaller set of simple instructions for improved performance.

[2] big.LITTLE is an innovative computing architecture developed by ARM, which combines high-performance (big) and energy-efficient (LITTLE) processor cores on the same chip. DynamIQ enhances this architecture by offering greater flexibility in core configurations.

capabilities, which support mesh shading and hardware-accelerated ray tracing, thereby pushing the boundaries of graphics rendering. On the other hand, the Qualcomm Snapdragon W5+ features co-processors specifically tailored for ML and AI tasks, underscoring the industry's commitment to crafting platforms adept at managing the intensive demands of data-driven applications and services. This progression towards a more heterogeneous and specialized computing landscape mirrors the escalating need to navigate the intricate computational requirements posed by modern applications.

Continuing the exploration of the shift towards specialized and heterogeneous computing frameworks, the development and prototyping of a resilient software architecture dedicated to massive sensor-based data processing underscores the critical need for versatile general-purpose platforms. These platforms are required to proficiently handle an extensive range of applications and software environments. Jeff McVeigh, the General Manager of the Super Compute Group at Intel, underscored the substantial advantages of integration—such as improved performance, reduced power consumption, and cost efficiency—in his keynote at ISC23 [122]. Yet, he also pointed out a fundamental compromise: the loss of flexibility. This observation highlights the dilemmas encountered in the "Falcon Shores" project, leading to a reevaluation of performance benchmarks. Driven by the imperative for enhanced adaptability, our research shifts focus towards the advantages presented by a heterogeneous assembly of computing devices. The integration of CPUs, iGPUs, dedicated GPU, and FPGAs into a singular heterogeneous platform is identified as the optimal solution. This system is ideal for efficiently processing massive amounts of data by combining high performance, scalability, and adaptability. The approach aligns with current technological trends and ensures that our architecture can meet diverse computational needs.

Heterogeneous computing architectures, pivotal in addressing the complexities of massive sensor-based data processing, can be broadly categorized into two main groups. The first group encompasses SoC, where the accelerator and the CPU are integrated on the same chip. This configuration promotes enhanced communication speeds between devices, albeit with performance limitations dictated by power consumption constraints, potentially leading to overheating and thermal throttling. The second group consists of architectures in which the accelerator is discrete and connected to the CPU via the motherboard's PCIe[3] interface. This arrangement offers increased performance and power for the accelerator but may compromise the communication speed with the CPU, risking bottlenecks in memory-intensive tasks. While both architectural strategies present viable path-

[3]PCI Express, or Peripheral Component Interconnect Express (PCIe), is a high-speed serial computer expansion bus standard, facilitating the connection of hardware devices to a computer.

ways for explaining chip architecture divergences, our discourse will concentrate on SoC architectures, although we also include some exploration of heterogeneous architectures with discrete accelerators. This focus enables a thorough examination of the performance challenges inherent in these systems and the industry-proposed solutions aimed at mitigating such issues. This discussion sets the stage for the forthcoming section, where we will explore the specifics of these critical components within the broader context of heterogeneous computing platforms.

### 2.3.2. CPU and GPU architectures

As a Soc example, Figure 2.4 displays the die of the Intel *Alder Lake S* SoC architecture, specifically the Core i9-12900k, one of the multiple processors used for experimental evaluations in the present work. It has dimensions of $208.8mm^2$ ($20.4mm \times 10.2mm$), Intel's 7-nm lithography, and a Thermal Design Power (TDP) of 125W. The SoC integrates a CPU, a GPU, and additional logic for I/O[4] and memory management. The CPU count with 16 cores divided into two types: 8 Performance-cores (P-cores), based on *Golden Cove* microarchitecture, and 8 Efficient-cores (E-cores), based on *Gracemont* microarchitecture. The GPU is based on the Intel *Xe-LP* architecture and count with 32 Execution Unit (EU), respectively 256 shader cores. All this logic is connected to the I/O and the memory management through a ring bus that allows shared access to the L3 or Last Level Cache (LLC), facilitating the communication between integrated devices. The platform also counts with 128 GB of DDR5 memory with a transfer rate of 4800 MT/s, divided into four modules of 32 GB each.

Operating at a base clock of 3.20 GHz, which can be boosted up to 5.2 GHz in turbo mode, the P-cores are tailored for high-performance tasks requiring low latency, such as gaming, content creation, and productivity. In contrast, the E-cores, which have a base clock of 2.40 GHz and can achieve up to 3.9 GHz in turbo mode, are optimized for power-constrained or multi-threaded scenarios, including web browsing, streaming, and background processes. The cache memory hierarchy consists of two private levels of cache per core: a 640 KB L1 Data Cache (L1d) and a 768 KB L1 Instruction Cache (L1i) divided into 16 instances (one per core), and a 14 MB L2 Unified Cache (L2), from which 1.25 MB are instantiated per P-core, and 2 MB are shared between four E-cores. The L3 cache is shared between all cores, totaling 30 MB, and is divided into 10 slices, each with 2 MB. The P-cores and E-cores can work together or independently, depending on the workload and power requirements. Figure2.4 also shows that the CPU supports DDR5 memory, allowing for memory transfer rates up to 4800 million

[4]Input/Output

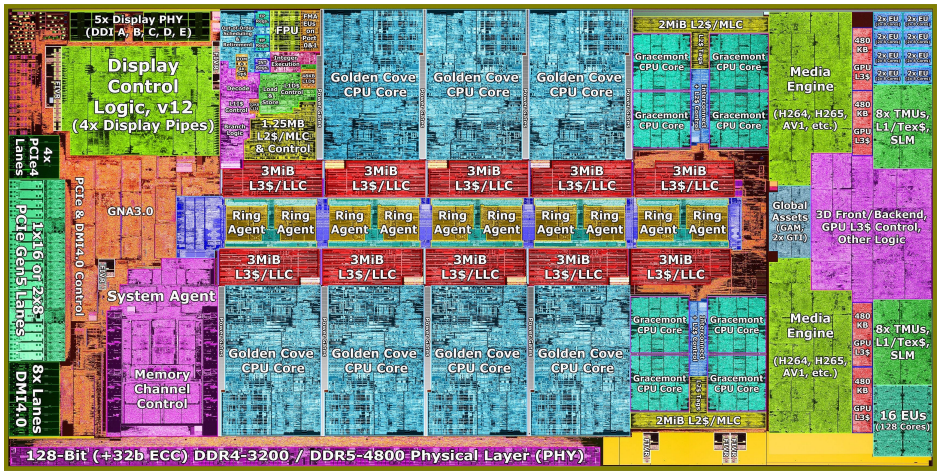



Figure 2.4: Alder Lake S

transfers per second (MT/s), and PCIe 5.0, which doubles the bandwidth of PCIe 4.0 achieving communication bandwidths of 3.94 gigabytes per second (GB/s) in both directions simultaneously, allowing for faster data transfer rates between the SoC and other accelerators. In order to dynamically assign threads to the appropriate cores based on the application and system state, the CPU also integrates a new *Thread Director* technology built into hardware and firmware that guides the operating system (OS) to assign threads to the appropriate cores. The CPU can support up to 16 cores (8 P-cores and 8 E-cores) and 24 threads (16 P-threads and 8 E-threads) using Hyper-Threading (HT) technology, which is not supported by the E-cores. The CPU follows a Complex Instruction Set Computer (CISC) architecture, which aims to reduce the number of instructions and cycles per program by utilizing a large number of complex instructions. This is in contrast to RISC architectures. Both cores have the same level of instruction support, Advanced Vector Extensions 2 (AVX2) yields 256-bit Single Instruction, Multiple Data (SIMD)[5] operations, and Vector Neural Network Instructions (VNNI) that accelerates convolutional neural network-based algorithms. The design of each core type has also been improved with a wider decoder in the front-end, increasing the ILP, a smarter branch predictor, which reduces the misprediction

[5]SIMD, is a parallel computing architecture that allows a single instruction to be simultaneously performed on multiple data points. This exploits data-level parallelism by processing data in vectors, significantly increasing throughput and efficiency for operations that can be executed in parallel, such as those found in multimedia processing, scientific simulations, and ML algorithms.

penalty and increases the performance, and more executions ports in the backend that allows for more ILP to be exploited.

Following our examination of CPU architectures and the strategic shift towards heterogeneous computing models, it is essential to highlight the evolution of graphics hardware throughout the 1990s and 2000s. Originally developed to meet the specialized requirements of the graphics industry, particularly in gaming, graphics hardware has experienced a significant acceleration in performance [24]. The rate of this advancement, exceeding an annual factor of 2.4, surpasses the predictions of Moore's law. This remarkable increase in performance is largely due to the hardware's capacity to leverage the parallelism inherent in computer graphics processing. Fueled by the insatiable demand for FLOPS and memory bandwidth from the gaming and graphics sectors, GPUs evolved into highly parallel, multithreaded, manycore processors, boasting significant computational power and extensive memory bandwidth. By the year 2001, the dominance of PC graphics signaled a decline in the era of large, dedicated graphics systems. This transition encouraged the developer community to utilize the powerful computational capabilities of graphics hardware for accelerating scientific workloads. Consequently, the application of GPU architectures extended far beyond their initial graphics-focused domain, illustrating their versatility and vital role in the broader landscape of scientific computing and data processing [24].


(a) Intel Xe-LP slice. (b) Intel Xe-LP EU.

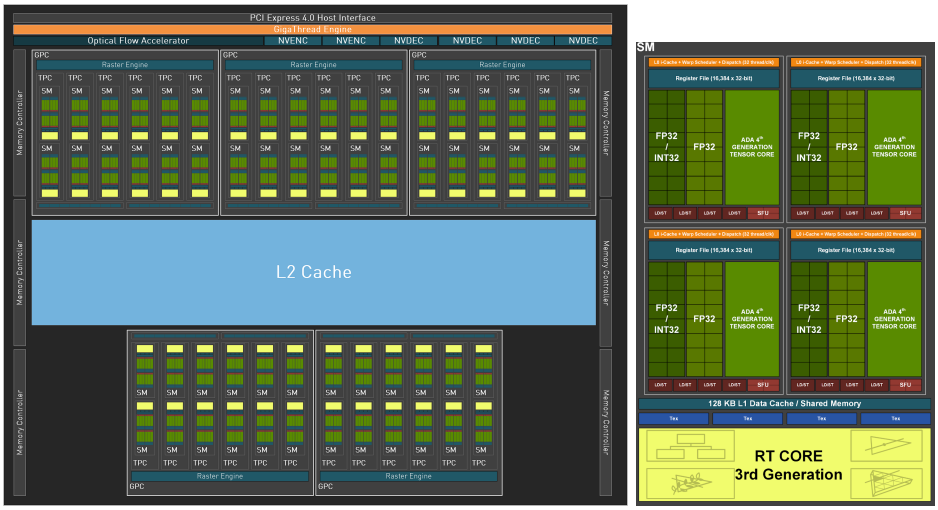
Figure 2.5: Intel Xe-LP block diagram.

In order to explore into the GPU hardware, we will first analyze the Intel Xe-LP architecture first, shown in Figure 2.5, which is integrated into the Alder Lake SoC. This GPU, known as the Intel UHD Graphics 770, has a base clock of 300 MHz, which can be boosted up to 1.55 GHz using Intel's Graphics Max

Dynamic Frequency technology. GPU architectures have a distinctive feature of considerable frequency disparity with the CPU, which poses a challenge for communication between integrated devices. The Intel Xe-LP architecture, is optimized for performance per watt with a 15 W TDP and designed for low-power applications. The iGPU architecture comprises two sub-slices, each containing 16 EUs, for a total of 32 EUs. These units span 33.5 $mm^2$ of the die area, which also accounts for the media engines. The EU is the fundamental unit of the GPU architecture, as depicted in Figure 2.5a. This figure illustrates the block diagram of the Intel Xe-LP slice of a larger GPU with 96 EUs, but it is useful for understanding the architecture as the rest of the logic remains the same. The EU integrates Simultaneous Multi-Threading (SMT) and fine-grained Interleaved Multi-Threading (IMT). The EUs are processors that incorporate multiple single instructions, multiple data arithmetic logic units, and SIMD Arithmetic Logic Units (ALUs). As Figure 2.5b shows, the EU consist of 8-wide (SIMD8) ALUs for floating-point and integer operations, and 2-wide (SIMD2) ALUs for extended math (EM) operations. Like the CPU, the iGPU also includes a new instruction for neural network inference called Data Product 4x8 (DP4A - 8-bit integer Dot-Product of 4 Elements and Accumulate). This instruction accelerates the execution of 8-bit integer (INT8) matrix multiplication operations to make up of dedicated hardware such as tensor cores. The processing units are organized in a pipeline across multiple threads, which enables high throughput for both integer and floating-point operations. Depending on the software workload, the threads within a processing unit can execute the same or different kernel code. Additionally, the fine-grained structure ensures a continuous flow of instructions. This structure allows the concealment of latency during memory scatter/gather or longer operations. One of the notable changes in this iGPU microarchitecture is that the EU is no longer a stand-alone block. Rather, two EUs now share a single thread control unit, making it easier to distribute the workload across the EUs and simplifying the thread scheduling hardware overall, as shown in Figure2.5b.

The distribution of memory can be analyzed from the bottom up. Each sub-slice has a shared local memory (SLM) that is located near the EUs. This proximity enhances efficiency and reduces latency of the EUs, improving the effectiveness rate for atomic operations. The local memory is separate from the L3 cache, which is shared by all sub-slices. Above them, the GPU and CPU share a physical Dynamic Random Access Memory (DRAM). This unified memory architecture provides power efficiency and programmability benefits compared to discrete memory transfers between devices and hosts or between hosts and devices through PCIe. The main advantage of this architecture is the zero-copy

buffer transfers between the CPU and GPU due to the shared physical memory. Performance is enhanced by adding an LLC shared with the main memory.




(a) NVIDIA AD104 GPU block diagram. (b) CUDA SM.

Figure 2.6: NVIDIA AD104 GPU architecture.

Going deeper into GPU architectures, we will explore the architecture of the NVIDIA AD104 chip, which is packaged as the GeForce RTX 4070 Ti and installed in one of our experimental heterogeneous platforms that also incorporates the *Alder Lake S* SoC previously discussed. The AD104 is a dedicated GPU, based on NVIDIA's *Ada Lovelace* architecture, named after Augusta Ada King, Countess of Lovelace, who is widely recognized as the first computer programmer. The AD104 features a die size of $295mm^2$, is built using TSMC's 4-nm lithography, has a 285 W TDP, and operates at a base clock of 2.3 GHz, which can be boosted up to 2.6 GHz. It also integrates 12 GB of GDDR6X memory, which provides a 504.2 GB/s memory bandwidth. The stark contrast between a dedicated GPU and an iGPU detailed above, becomes evident when considering these physical characteristics. Figure 2.6 presents the block diagram of the AD104 chip on the left, and what the CUDA specification terms as the Streaming Multiprocessor (SM) on the right. As illustrated in Figure 2.6a, NVIDIA's GPU architecture is notably hierarchical and symmetric, demonstrating various hardware levels. This work does not provide an exhaustive analysis of all the specifics, and for a more comprehensive explanation, the reader is referred to the *Ada Lovelace* architecture whitepaper. However, some details about the funda-

mental units of the architecture are worth mentioning. The Graphic Processing Cluster (GPC) is the main high-level hardware block, comprising various types of processing units. In the AD104 chip, there are 5 GPCs, each with 12 SMs, making a total of 60 SMs. The Processing Block (PB), the basic computational block in NVIDIA's GPU architectures, can be found in a cluster of 4 PBs within each SM. This case illustrates too the stark contrast between the iGPU and the dedicated GPU, as the AD104 features 240 PBs, while the iGPU has 32 EUs, when comparing fundamental blocks of both architectures. As depicted in Figure 2.6b, the *Ada Lovelace* SM comprises several key components. Each PB features a *warp scheduler*, a 32-thread wide scheduler that is an essential element in the CUDA environment. The PBs also encapsulate 16 CUDA cores dedicated to FP32 operations, and an additional 16 CUDA cores that can handle either FP32 or INT32 operations. This configuration allows each PB to process up to 32 FP32 operations per cycle in the best-case scenario. To achieve peak performance, it is crucial that the *warp scheduler* maps the 32 threads to the 32 CUDA cores. From a programming perspective, the concepts of maintaining *memory coalescence* to fully utilize the memory bus, and minimizing *warp divergence* to engage all CUDA cores in the SM, are crucial strategies. These concepts are instrumental in achieving high performance in CUDA-capable GPUs.

The intricate architecture of the AD104 chip, with its advanced processing units and the *Ada Lovelace* SM design, marks a pivotal moment in the evolution of discrete GPU architectures. The shift towards incorporating a higher number of specialized processing blocks exemplifies the industry's push towards optimizing computational efficiency and power. This emphasis on specialized hardware for specific computational tasks reflects a broader trend in the evolution of computing hardware towards greater specialization and heterogeneity. As we explore the implications of these hardware advancements, it becomes evident that the distinction between integrated and discrete GPU architectures extends beyond mere performance metrics; it is indicative of the underlying architectural philosophies that prioritize efficiency, parallelism, and task-specific optimization. This evolution in hardware design, characterized by a move towards more complex and specialized components, illustrated with recent releases of the Universal Chiplet Interconnect Express (UCIe) new specification for advanced chiplet architectures, or the Chip-on-Wafer-on-Substrate (CoWoS) TSMC's proprietary technology that enables system integration of multiple chips at wafer level, lays the groundwork for a subsequent shift in focus towards software development and the integration of complex embedded software within SoC and SiP configurations.

The transition from hardware evolution to software development is not merely a shift in focus but a necessary progression to harness the full potential of these

advanced hardware architectures. As computing frameworks become increasingly heterogeneous and specialized, the role of software in effectively managing and leveraging this complexity becomes paramount. The integration of complex embedded software with SoC and SiP configurations, highlights the growing interdependence between hardware and software, underscoring the need for innovative programming models and languages that can navigate the intricacies of these advanced systems. Moreover, the MtM initiative seeks to incorporate non-digital features into silicon-based technologies, which represents a significant leap forward in extending the capabilities of silicon-based technologies. This endeavor expands upon the accomplishments of the "More Moore" advancements and creates new opportunities for system-of-systems frameworks. Frameworks like these represent a significant milestone in the evolution of software architecture, which is closely intertwined with hardware advancements. The convergence of digital and non-digital technologies within a unified system architecture highlights the crucial role of software in improving the performance and functionality of modern computing systems. This marks a pivotal chapter in the ongoing evolution of computing technology.

## 2.4. Software evolution

Software engineering has witnessed profound changes over several decades, shaped by diverse influences including usability, performance and security. This journey has seen software engineering emerge as a distinct discipline, initially centered on mainframe computing. This early phase was characterized by operations within isolated, non-networked environments, making it accessible to a limited audience and somewhat removed from real-world applications. The formalization of "software engineering" as a term by Margaret Hamilton in the 1970s, through her pioneering work on the Apollo XI mission's navigation system [57], was a defining moment, highlighting the complexity and crucial role of software development. As illustrated with the Figure 2.7, the timeline from the late 1990s into the early 2000s reflects a period where foundational methods were being established, with an emphasis on usability and knowledge management systems, progressing to address the human aspects and adoption practices within the field.

Entering the 21st century, software engineering experienced a dramatic shift, paralleled by the advancements indicated in the timeline in Figure 2.7. As the new millennium commenced, the focus expanded to include large-scale agile methodologies and the integration of security considerations into global software engineering practices, accommodating the complexity of architecture in safety-critical

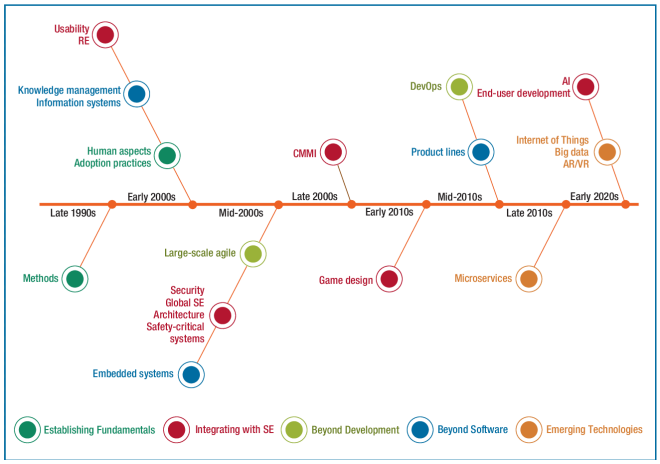



Figure 2.7: 50 years of software history [48].

systems. This era introduced programming models like MPI and OpenMP, conceived to optimize the performance of HPC systems. Concurrently, the explosion of the internet, as denoted by the significant marker in the timeline during the mid-2000s, facilitated the creation of web-based applications and services. This propelled the field into new paradigms such as Capability Maturity Model Integration (CMMI) and game design in the late 2000s and the adoption of DevOps (Software Development and IT Operations) practices by the early 2010s, necessitating innovative frameworks in software development to meet the demands of an increasingly interconnected world. As illustrated, the late 2010s saw the rise of microservices architecture, setting the stage for the latest advancements in AI end-user development, Internet of Things, big data, and AR/Virtual Reality (VR) technologies in the early 2020s, which continue to redefine the landscape of software engineering.

The pivot towards energy efficiency, highlighted in Section 2.2, has catalyzed the development of specialized heterogeneous architectures that integrate CPUs and other devices within a single SoC, or connect CPUs to discrete accelerators, as detailed in Section 2.3.1. This evolution underscores the necessity for programming models that leverage the distinct capabilities of these complex systems. In the realm of general-purpose computing, milestones such as the introduction of CUDA, OpenCL, and SYCL represent significant advances in programming models. Yet, the advent of these models has also brought to light challenges related to portability and interoperability across emerging hardware architectures. These difficulties stem from the models' inability to consistently achieve high levels of

performance while facilitating programmer productivity on these novel architectures. In response, comprehensive Application Programming Interfaces (APIs) and frameworks, notably Intel's oneAPI [59], have been developed. These aim to standardize the programming approach across different architectures through a unified language, addressing the critical need for a cohesive and efficient development environment in the face of increasingly heterogeneous computing landscapes, paving the way for higher levels of abstraction in programming without sacrificing performance or the ability to fine-tune code to the intricacies of the underlying hardware.

### 2.4.1. SYCL

Building on the foundation of these frameworks, SYCL emerges as a robust cross-platform abstraction layer within the Khronos Group's royalty-free open standard, enabling developers to write standard C++ code for heterogeneous processors. Both the host and device code for an application are contained in the same source file. SYCL [121] leverages generic programming, employing templates and generic lambda functions, to facilitate the development of higher-level application software. This approach allows for the clean coding of applications with optimized acceleration of kernel code across a variety of acceleration APIs, including OpenCL. It enables developers to operate at a more abstract level compared to native acceleration APIs, while still providing the option to access lower-level code for detailed optimization. This dual capability is made possible through interoperability mode, alongside the use of standard C/C++ libraries and frameworks such as OpenCV or OpenMP, ensuring a seamless integration with native acceleration APIs. The ecosystem of SYCL implementations is experiencing robust growth, bolstered by contributions from an increasing number of vendors. These implementations are designed to support an extensive assortment of acceleration API back-ends, not limited to OpenCL. A key objective of SYCL is to enable the targeting of any accelerator, aptly encompassed by Intel's newly coined term "XPU" [6]. This term refers to a range of hardware accelerators, including CPUs, GPUs, FPGAs, and other potential future computing platforms. SYCL's intent to provide an abstraction layer over these various forms of hardware accelerators is pivotal, as it allows developers to write code that is agnostic to the underlying accelerator technology. This abstraction is instrumental in harnessing the full potential of diverse hardware architectures, offering a versatile and unified programming model that can adapt to the ever-evolving landscape of computational accelerators. The rising popularity of SYCL has led to diverse implementations, each with unique capabilities, as depicted in Figure 2.8:

- **Intel oneAPI** [59]: Intel's oneAPI ecosystem implements SYCL to support development across Intel CPUs, GPUs, and FPGAs. It encompasses additional tools and libraries to ease the adoption of SYCL and leverage the capabilities of heterogeneous programming.
- **ComputeCpp** by Codeplay: It was discontinued in July 2023, since when the company has been focusing on upstreaming its SYCL implementation to the Intel's LLVM project [60]. This was a well-established implementation known for its support of a variety of OpenCL-compliant devices from multiple vendors. ComputeCpp offered advanced features that aid in the development process across different accelerators, enhancing productivity and support for developers.
- **triSYCL**: What began as an open-source project at AMD is now under Xilinx's stewardship. triSYCL is a work-in-progress implementation of the SYCL specification that utilizes modern C++20 features. It focuses on Xilinx's FPGA architectures and the Versal ACAP while maintaining compatibility with CPUs via OpenMP and other accelerators through OpenCL, SPIR, or LLVM. Collaboration with oneAPI's SYCL implementation is ongoing to broaden its reach [136].
- **AdaptiveCpp** (formerly known as hipSYCL): Originating from the University of Heidelberg, AdaptiveCpp bypasses OpenCL in favor of compiler toolchains like CUDA for NVIDIA devices and HIP for AMD's ROCm stack compatibility [127]. It is also capable of supporting CPUs through OpenMP and is exploring support for Intel GPUs with oneAPI Level Zero and SPIR-V, though this remains in an experimental phase.
- **neoSYCL**: A product of Tohoku University's development efforts [126], neoSYCL is unique in its support for NEC's SX-Aurora TSUBASA architecture [73], using its vector engine for application acceleration. Based on the LLVM and Clang infrastructure, neoSYCL facilitates the use of C++17 Standard Template Library (STL) while currently omitting support for some OpenCL-specific features.

Each implementation contributes to the versatility of SYCL, reflecting the standard's capacity to meet the demands for efficient, high-level programming across a diverse range of computing architectures, thereby enhancing the potential for innovation in the field of accelerated computing.

This work examines Intel oneAPI [59], a framework that simplifies programming heterogeneous architectures by providing several libraries, tools, and a unified programming language, C++ with SYCL. SYCL is enriched with extensions such as Unified Shared Memory (USM), ordered queues, reductions, subgroups (applicable to both CPU and GPU implementations), and data flow

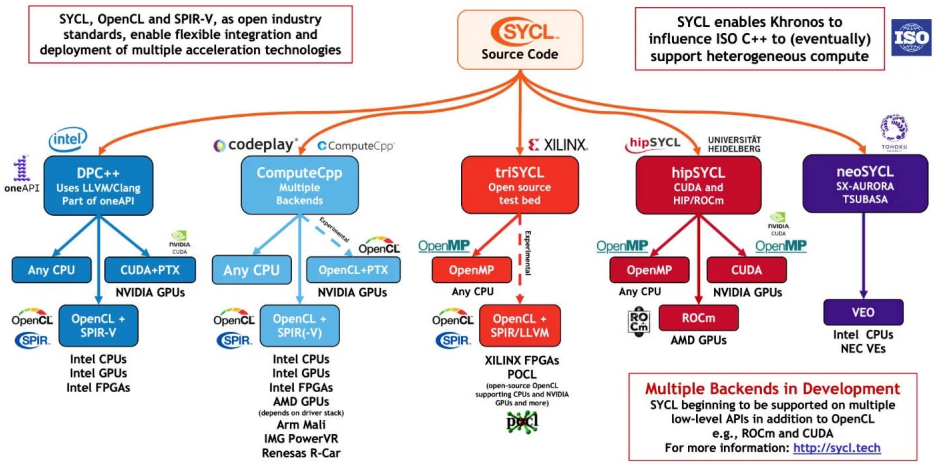



Figure 2.8: SYCL implementations.

pipes (specifically designed for FPGA support). A pivotal advantage of adopting the oneAPI framework with SYCL, in contrast to specific programming languages such as OpenCL or CUDA, lies in its unified programming language framework. This singular approach facilitates the targeting of diverse platforms using the same programming model, which streamlines the development process, enhancing code portability, simplicity, and maintainability. Both host and accelerator codes can be seamlessly integrated within a single source file, employing standard C++ syntax for the articulation of parallelism without the need for introducing new keywords or pragmas. Parallel constructs are encapsulated within C++ classes, allowing for intuitive expression of parallelism. The selection of the target accelerator, whether it be a CPU, GPU, or FPGA, is managed through the device `queue` within the host code, which is initialized with various flags for device selection: (1) `default_selector`, which employs heuristics to determine the most suitable device among those available; (2) `cpu_selector`, for selecting the CPU; (3) `gpu_selector`, for GPU targeting; (4) `ext::intel::fpga_emulator_selector`, which targets the CPU as an emulator for Intel's FPGA; (5) `ext::intel::fpga_selector`, for deploying on an actual Intel FPGA; and (6) `accelerator_selector`, customizable for targeting other generic devices. This flexibility underscores the adaptability of SYCL in accommodating a wide range of computational resources.

In programming environments where offload computation is essential, compilers are responsible for generating code tailored for both the host and the device

platforms. The SYCL programming model introduces the concept of single-source compilation, which offers several benefits over the traditional approach of compiling host and device code separately. Notably, single-source compilation enhances usability by reducing the number of files developers need to manage, allowing for the direct inclusion of device code within the vicinity of its corresponding host code call site. This model streamlines the development workflow and fosters a more integrated coding environment improving programmer productivity. Employing a single compiler for both host and device code compilation also introduces additional safety measures. By overseeing the interface between host and device code, the compiler ensures that the actual parameters passed by the host match the formal parameters expected by the device kernel. This cross-checking mechanism mitigates the risk of mismatches and potential errors. Furthermore, the unified compilation process enables the device compiler to perform further optimizations by leveraging the broader context of kernel invocation. Such optimizations may include constant propagation and improved inference of pointer aliasing across function calls, thereby enhancing the overall efficiency and performance of the application. The SYCL programming model, while supporting separate compilation, emphasizes the advantages of the single-source approach, aligning with the goal of simplifying the development process for heterogeneous computing environments. While SYCL applications are versatile enough to run on any supported target hardware, achieving optimal performance necessitates tuning for the specific architecture in question. In SYCL, any device, including the CPU, can be selected as a target accelerator through the device queue mechanism (e.g., using `cpu_selector`), effectively treating the CPU as an offload target similar to other accelerators. However, code that is optimized for GPU execution, such as algorithms designed to exploit massive thread parallelism, may not achieve the same level of performance when run on a CPU serving as an accelerator, due to fundamental architectural differences in how these devices handle parallel workloads. This highlights the importance of architecture-specific optimization in the development process.

The Figure 2.9 depicts the compilation and execution workflow for SYCL. In this workflow, SYCL acts as an intermediary layer that enables the development of applications capable of running on a variety of hardware platforms. Starting with C++ libraries, developers can write standard application code or leverage ML frameworks like TensorFlow. This code can utilize C++ template libraries for further abstraction and flexibility. The SYCL compiler plays a crucial role by processing the C++ code, which includes both host and device computations described through templates and lambda functions. This code is then compiled into an intermediate form that is suitable for execution on different types of hard-

ware accelerators, such as CPUs, GPUs, FPGAs, DSPs, AI/Tensor hardware, or custom hardware platforms. If the target is a CPU, the code may also go through a standard CPU compiler like LLVM, GCC, or Visual C++. For hardware accelerators, the SYCL compiler, which is built on top of the LLVM infrastructure, interacts with OpenCL, allowing the generated code to be further compiled by device-specific OpenCL compilers. This step ensures that the accelerated code can run optimally on the target device. The Figure 2.9 underscores that SYCL is optimized for performance portability across different hardware, making it an ideal choice for developing large-scale C++-based engines and applications that require efficient execution across diverse computing environments. Additionally, SYCL's ability to work with kernel fusion suggests it can offer better performance on complex applications and libraries compared to traditional hand-coding techniques.

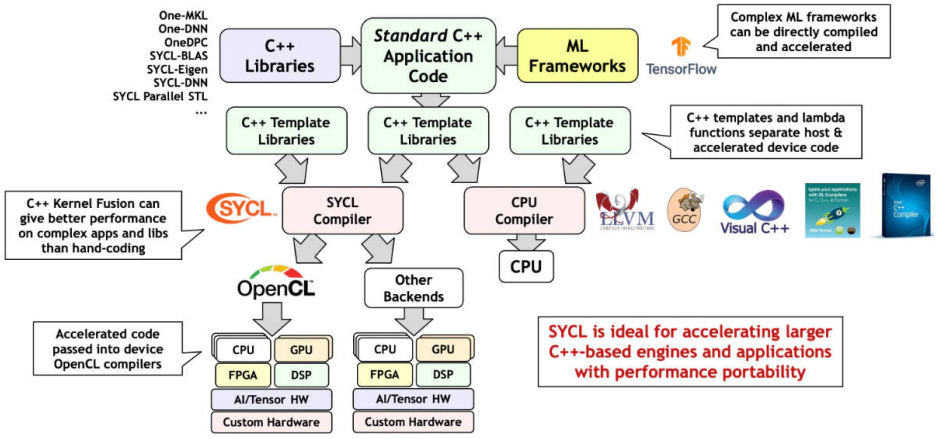



Figure 2.9: SYCL compilation flow.

Utilizing the Intel oneAPI C++ Compiler (ICX) driver [58], `icpx`, to compile C++ files, along with the `-fsycl` flag which invokes ICX compiler with SYCL extensions, applications are compiled in a way that produces the required code for execution on both host and device. When writing C++ applications with SYCL, the compilation workflow is facilitated by these drivers and offers two distinct pathways for device code generation: Just-in-Time (JIT) and Ahead-of-Time (AOT) compilation, which can be leveraged according to the specific requirements of the development and deployment environments. By default, the JIT compilation approach enables runtime determination of available devices, facilitating the generation of device-specific code dynamically. This approach en-

sures adaptability, albeit with potential performance drawbacks due to runtime compilation. In contrast, AOT compilation demands explicit device specification during the compile-time, potentially enhancing performance by eliminating runtime compilation delays. However, this method's efficiency is contingent upon the application's scale and the volume of device code, as larger applications may suffer performance degradation from extended compilation times. It is pertinent to highlight the unique compilation requirements for FPGAs compared to CPUs and GPUs. FPGA device code generation is notably resource-intensive and prolonged, typically spanning several hours. This characteristic renders JIT compilation for FPGAs impractical, necessitating reliance on AOT compilation exclusively. To mitigate these challenges, the ICX compiler introduces various facilitative mechanisms, such as CPU emulation of FPGAs, partial compilation, and simulation techniques. These tools are designed to expedite the FPGA design process, allowing for rapid prototyping and iteration [103].

In the context of this thesis, SYCL and the Intel oneAPI framework play a pivotal role in the experiments detailed in Chapter 3. Specifically, these technologies were used to efficiently offload computations to various accelerators, taking full advantage of the heterogeneous computing environments available. The `queue` feature of SYCL was instrumental in this process, allowing for task distribution across different hardware architectures. By utilizing SYCL's abstraction capabilities, the LiDAR data processing algorithms were implemented in a manner that maintained portability while exploiting the strengths of each target accelerator. This approach demonstrated the versatility of SYCL in handling massive sensor-based analytics, and showcased its potential in optimizing performance across diverse computational platforms (CPUs, discrete GPUs, and iGPUs) from different vendors (Intel and NVIDIA). The implementation aligns with the work's focus on accelerating analytics on heterogeneous architectures, providing a practical demonstration of SYCL's capabilities in a multi-vendor, multi-architecture environment.

### 2.4.2. CUDA

CUDA, standing for Compute Unified Device Architecture, is a parallel computing platform and API model, developed by NVIDIA, that allows to utilize CUDA-compatible devices for general-purpose computing. A simplified representation of the comprehensive CUDA environment is depicted in Figure 2.10, offering a visual overview of the available resources to developers within this advanced computational framework. As illustrated in the figure, the CUDA ecosystem provides a comprehensive suite of tools and libraries for GPU-accelerated

computing. At its core, CUDA is a parallel computing platform that leverages NVIDIA's GPU architecture to enhance computing performance across a range of applications, from consumer internet services to supercomputing. CUDA's capabilities are accessible via a rich set of proprietary libraries, such as cuBLAS and cuDNN, optimized for specific tasks like linear algebra and deep learning. The robustness of the platform is further supported by the CUDA Toolkit, which includes a dedicated compiler (called NVCC, which is built on top of the LLVM compiler infrastructure), development tools, and language extensions for C/C++, Python, and Fortran, allowing for integration with OS platforms and customization through memory management and graphics libraries via the CUDA Driver, as shown in the lower part of Figure 2.10 from the CUDA layer of the environment stack. However, CUDA's environment, while robust, is limited by its proprietary nature and exclusive optimization for NVIDIA GPUs, lacking support for other proprietary hardware architectures. These limitations are addressed by SYCL, as discussed in Section 2.4.1, presenting a cross-platform alternative for developers, facilitating heterogeneous computing across a variety of accelerators. Unlike CUDA's proprietary nature and optimization for NVIDIA hardware, SYCL aims for hardware agnosticism through an open standard approach. While CUDA offers an extensive set of libraries finely tuned for NVIDIA's GPUs, SYCL relies on a growing ecosystem of cross-platform libraries that follow the SYCL standard. Although both environments provide a comprehensive set of tools for developing high-performance applications, SYCL's emphasis on cross-platform support positions it as a versatile choice for developers targeting a broader range of hardware architectures.

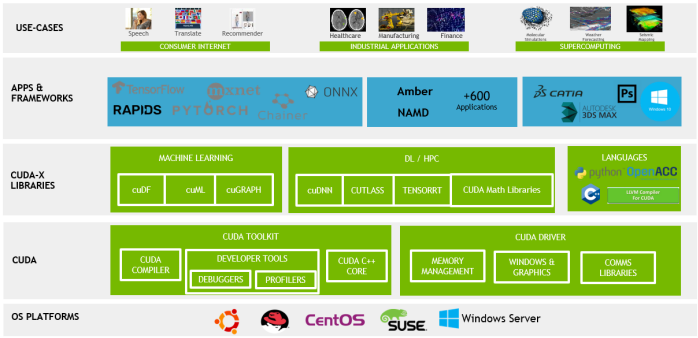



Figure 2.10: CUDA environment.

Originally tailored as a proprietary framework for NVIDIA GPUs, the CUDA platform has evolved to embrace a broader spectrum of architectures, including

x86 and ARM. This expansion has positioned CUDA as a favored tool among developers for crafting solutions that leverage heterogeneous CPU+GPU systems, highlighting the broader software evolution explored in Section 2.4. The platform's applicability has further widened to encompass low-power devices and edge accelerators, such as the NVIDIA Jetson series, underlining some of the opportunities identified in Section 2.2. Notably, NVIDIA's acquisition of ARM catalyzed the integration of ARM-based technologies into its lineup, exemplified by the NVIDIA Grace-Hopper SoC and the NVIDIA Bluefield-3 Data Processing Unit (DPU), underscoring NVIDIA's commitment to diversifying its portfolio of computing solutions. Despite the broad utility of CUDA across various architectures, this discussion specifically focuses on its application within NVIDIA GPU environments, underscoring the platform's pivotal role in driving advancements in GPU-accelerated computing.

The CUDA programming model is based on the concept of a *kernel*, the same as in SYCL. CUDA also uses a single-source compilation model, which allows the host and device code to be combined in a single source file. The CUDA threads are organized into a *grid*, which is further divided into *blocks*. Each *block* is directly mapped to a SM, and the threads within a *block* can cooperate by sharing data through shared memory. Recovering the discussion about the *warp scheduler* in Section 2.3.2, this gives us a hint about the best size for a *block* for this architecture, as it should be a multiple of 128 (32 threads/PB $\times$ 4 PBs/SM), in order to fully utilize the SM. NVIDIA GPUs exploit fine-grain and massive data parallelism, thus allowing for the use of as many *blocks* as the application requires. However, to achieve optimal performance on any NVIDIA GPU architecture, the number of threads per *block* must remain a multiple of "warp-width $\times$ the number of PBs per SM". This is a crucial aspect of the CUDA programming model, as it directly impacts the performance of the application. To orchestrate the execution of the *kernel*, synchronization is provided at various levels, including the *kernel*, *grid*, *block*, or *thread* level. The *grid* is the highest level of parallelism in the CUDA model, and it is the responsibility of the developer to ensure that the *grid* is large enough to fully utilize the GPU's computational resources. The CUDA API provides synchronization mechanisms to coordinate the execution of *blocks* within a *grid*, and the execution of *grids* on the GPU (*kernel* level). This synchronization is essential to ensure that the GPU is fully utilized, and that the execution of the *grid* is completed in a timely manner.

The CUDA API delineates several abstraction layers, offering developers a spectrum of control and complexity. Notably, the driver and runtime APIs, while largely interchangeable, present distinct differences in terms of complexity and control. As illustrated in Figure 2.10, the runtime API, often synonymous with

the CUDA toolkit, streamlines the management of device code, albeit at the expense of the nuanced control available through the driver API. The latter affords meticulous control, particularly in the realms of context and module management, and boasts language independence. Context management, a capability intrinsic to the driver API, remains obfuscated within the runtime API, which autonomously ascertains the context pertinent to a thread. This delineation underscores the strategic choice between ease of use and the granularity of control, pivotal in harnessing the full potential of CUDA-enabled architectures.

In the context of this work, CUDA was used as the performance gold standard against which the SYCL implementations are compared. Leveraging CUDA's architecture-specific optimizations, the implementation exploits a range of fine-grained performance features to achieve peak efficiency on NVIDIA GPUs. These optimizations include the strategic use of shared memory within blocks (mapped to SMs), extensive register utilization, loop unrolling techniques, and ensuring memory coalescing for optimal memory access patterns. Furthermore, the implementation takes advantage of CUDA's block-level thread synchronization capabilities to coordinate parallel execution effectively. By these CUDA-specific features, the kernel implementations represent a best-case scenario for GPU performance, providing a benchmark for assessing the efficiency of the SYCL-based solutions across various architectures. This approach demonstrates CUDA's capabilities, and offers valuable insights into the performance portability of SYCL when targeting high-performance NVIDIA GPUs, as detailed in Chapter 3.

### 2.4.3. oneTBB

The oneAPI framework introduces a diverse collection of optimized libraries that streamline API-based programming for a variety of computing architectures. These libraries are optimized for compatibility with any supported target architecture, thereby negating the requirement for developer-led tuning efforts. For example, the BLAS routine from the Intel oneAPI Math Kernel Library (oneMKL) is finely optimized for both GPU and CPU applications. By employing library functionalities, API-based programming enables efficient device offload, saving developers considerable time in application development. The general recommendation is to initiate development with API-based programming, turning to SYCL offload features for scenarios where the API-based methods do not suffice. Within the extensive Intel oneAPI suite, developers have access to a wide range of libraries tailored for specific computational needs. These include oneCCL for collective communications, oneDAL for data analytics, oneDNN for deep neural networks, oneDPL for the Data Parallel C++ library STL, oneMKL for mathe-

matical kernels, oneVPL for video processing, and oneTBB for oneAPI Threading Building Blocks. This ensemble of libraries equips developers with powerful tools to optimize application performance across different computing platforms, echoing the versatile and comprehensive approach to heterogeneous computing.

Building upon the comprehensive suite of oneAPI libraries, oneTBB stands out as a pivotal C++ framework for crafting parallel applications [131]. This library addresses the gaps in parallelism support within the C++ standard, offering solutions in scenarios where existing language features are either inadequate or not universally supported across compilers. oneTBB extends the parallel programming capabilities of C++ by providing high-level abstractions that surpass what is currently or expected to be available in the C++ language standard.

Among its numerous features, oneTBB facilitates the development of scalable parallel applications by abstracting the complexity of direct thread management and synchronization. These features can be categorized into two main groups: parallel execution interfaces and execution model independent interfaces, as shown in Figure 2.11. The high-level execution interfaces, shown on the left side of the Figure 2.11, allow developers to define parallel tasks without concern for the intricacies of the thread lifecycle or inter-thread communication. With constructs such as the Flow Graph, oneTBB allows for the definition of complex data dependencies and asynchronous task execution, promoting a model where parallelism is derived from the data flow rather than explicit thread management. The integration of Parallel STL further eases the transition to parallel applications by extending the familiar STL interface to leverage the underlying parallel execution capabilities of oneTBB.

In addition, oneTBB's architecture is designed to scale with hardware, providing thread-safe concurrent containers, independent of the execution model, for efficient data management in a multi-threaded environment, as shown on the right side of the Figure 2.11. The library's memory allocation interfaces, such as the Scalable Allocator and the Cache Aligned Allocator, optimize dynamic memory operations that are critical for high-performance parallel computing. In addition, oneTBB provides a suite of synchronization primitives and utilities that facilitate fine-grained control over thread execution and access to shared resources. The provision of thread local storage ensures that individual threads have their own instance of data, eliminating unnecessary contention and enabling more efficient data handling in parallel environments. By encompassing these various aspects, oneTBB effectively manages threads by abstracting low-level details and providing a rich toolkit that promotes scalability, ease-of-use, and performance in parallel applications. This approach aligns with the objectives outlined throughout Section 2.4, emphasizing the importance of leveraging advanced libraries and

tools to exploit the full potential of heterogeneous computing architectures.

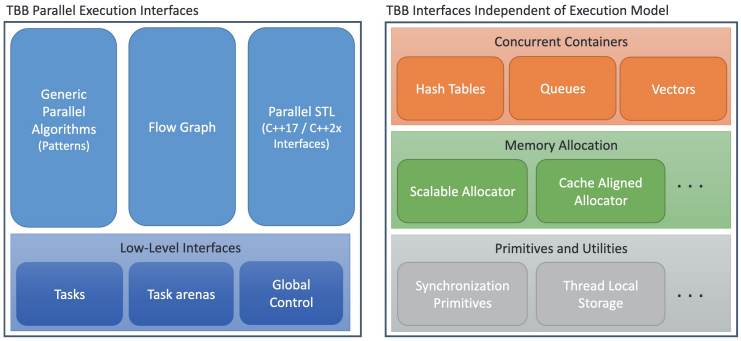



Figure 2.11: oneTBB library features [131].

Continuing the exploration of oneTBB's capabilities, it is crucial to understand the various layers of parallelism that parallel applications typically employ: the message-driven layer, the fork-join layer, and the SIMD layer [131]. The correspondence between these layers and the high-level parallelization interfaces of oneTBB is shown in Figure 2.12. oneTBB offers comprehensive support for the first two layers, while the third layer is typically addressed by the underlying hardware architecture and the compiler, although oneTBB does provide implementations of the SIMD layer for certain operations. The message-driven layer is characterized by large computations that interact through explicit messaging, encompassing patterns such as streaming graphs, data flow graphs, and dependency graphs. oneTBB adeptly facilitates these patterns through its Flow Graph interface, demonstrating its versatility in managing complex parallel workflows. The fork-join layer is critical for scenarios where sequential tasks branch into parallel executions, subsequently and then converge back to continue serially. This layer accommodates various programming constructs, including task parallelism, parallel loops, parallel reductions, and pipelines, all of which are efficiently supported by oneTBB's Generic Parallel Algorithms. This capability underscores oneTBB's strength in simplifying the decomposition and synchronization of parallel tasks, thereby improving application performance and scalability. Finally, the SIMD layer in parallel applications exploits data parallelism by performing identical operations on multiple data points simultaneously. This parallelism is typically exploited through vector extensions such as AVX, AVX2, and AVX-512, which were discussed in Section 2.3.2 as a key feature of modern computing architectures. The SIMD layer is responsible for optimizing computational efficiency for data-intensive tasks while exploiting the vector implementations present in all of

the oneTBB distributions that take advantage of the AVX extensions. Through these layers, oneTBB provides a comprehensive framework for expressing and executing parallelism that aligns with the advanced parallel programming paradigms discussed previously. This integrated support across multiple parallelism layers underscores oneTBB's role in enabling developers to create highly optimized and scalable parallel applications.

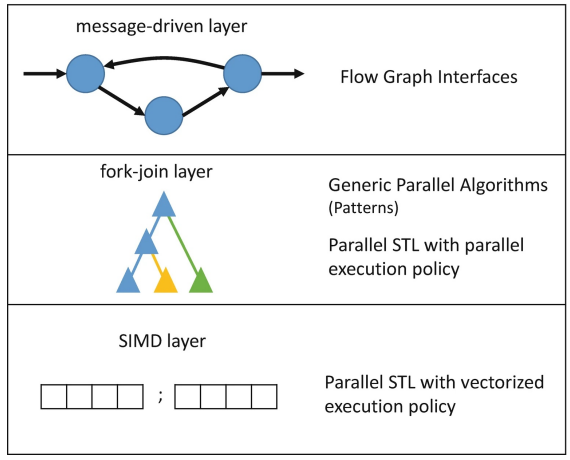



Figure 2.12: The three layers of application parallelism and their mapping to the high-level parallelization interfaces of TBB [131].

The full potential of oneTBB is realized through the principle of "composability", a powerful feature enabling the seamless integration of different parallel programming interfaces. For instance, an application might leverage the Flow Graph interface as its foundational layer. Nodes within this model could encapsulate nested Generic Parallel Algorithms, potentially enhanced with SIMD operations to exploit data parallelism to its fullest extent. This hierarchical structuring of parallelism, from high-level flow control down to low-level data operations, allows the oneTBB library to adeptly orchestrate task scheduling across the computing platform's resources. The "Task Scheduler" is the core of oneTBB [131]. It plays a crucial role in orchestrating the execution of new spawning tasks by dynamically assigning them to available threads. This system relies on a "work stealing" strategy to ensure a balanced distribution of tasks across all threads, thereby optimizing the use of processor resources. In this model, threads that have completed their assigned tasks actively seek out and "steal" work from busier threads, preventing any thread from remaining idle. This adaptive scheduling approach is particularly effective in dynamic environments where tasks vary in workload and

number and are spawned at runtime. Unlike "work-sharing" schedulers, which distribute tasks statically at spawn time, work stealing is a flexible and responsive strategy that allows for real-time load balancing, improving overall system performance and efficiency in multi-threaded scenarios.

The composability within oneTBB, highlighted by the nested integration of parallel programming interfaces, culminates in an efficient execution model managed by the "Task Scheduler". This synergy allows applications to achieve peak performance by fully leveraging the underlying hardware's capabilities, a concept that resonates with the broader themes of efficient resource utilization and parallel programming models discussed throughout Section 2.4. The Task Scheduler's use of a "work stealing" strategy is a testament to oneTBB's dynamic adaptability, which not only orchestrates the immediate scheduling of tasks but also ensures their optimal distribution in real-time across the processor's threads, embodying the library's core principle of composability.

In the context of this work, Intel oneTBB plays a crucial role in optimizing the CPU-based computations, particularly in Chapter 3. The library's advanced parallel programming features were leveraged to enhance the performance of two key stages: tree construction and tree traversal. During the tree construction phase, oneTBB's parallel algorithms and concurrent containers were employed to efficiently distribute the workload across multiple CPU cores, accelerating the process of building the spatial data structure. For the tree traversal stage, the library's task scheduler and work-stealing features were instrumental in balancing the computational load, especially when dealing with uneven tree depths and varying node densities. By utilizing oneTBB's composable parallelism model, the implementation could integrate different levels of parallelism, from high-level task parallelism in the overall algorithm structure to fine-grained data parallelism within individual tree operations. This approach improved the performance of the CPU-based LiDAR processing pipeline, and demonstrated the library's effectiveness in handling complex, hierarchical parallel computations in sensor-based analytics. The use of oneTBB in this work showcases its potential in accelerating data-intensive applications on multi-core CPU architectures.

## 2.5. Massive Sensor-Based Data Analytics

The previous discussion emphasized the critical role of developing new hardware architectures in effectively handling large amounts of data. Building upon this foundation, it is crucial to consider the wide range of applications enabled by these technological advances in massive sensor-based data analytics. Data

analytics is a field that involves uncovering trends, patterns, and correlations in large amounts of raw data to inform decision-making. This field employs statistical analysis techniques, such as clustering and regression, which are adapted for extensive datasets with the aid of modern tools. Since the early 2000s, advancements in software and hardware capabilities have enabled organizations to process large volumes of unstructured data, a trend further accelerated by the proliferation of technologies discussed in Section 2.1. This surge in data has necessitated the creation of innovative projects for data storage and processing, underscoring the field's continuous evolution as engineers seek novel ways to manage the complex information generated by sensors, networks, transactions, smart devices, web usage, and more.

The synergy between cutting-edge hardware architectures and sophisticated analytics algorithms has catalyzed transformative breakthroughs across diverse fields. In healthcare, for instance, the integration of advanced analytics with modern hardware architectures has had a significant impact on the analysis of large-scale genomic datasets, leading to unprecedented insights into genetic disorders and facilitating personalized medicine. In the field of urban informatics, the real-time processing of data from extensive networks of sensors and devices, empowered by these innovations, can enable smarter city management, enhancing efficiency in transportation, energy use, and emergency services. Similarly, in environmental science, the ability to process and analyze extensive datasets has been pivotal in modeling climate change scenarios with unprecedented precision [83], while in finance, the ability to process and analyze large volumes of financial data from various sources has been instrumental in developing sophisticated predictive models and risk management strategies. In this thesis we focus on applications that require processing large-scale data collections from sensors and are subject to real time and low power consumption constraints. These applications underscore the significance of the hardware innovations previously discussed in Section 2.3, and emphasize the need for a continuous evolution of computational paradigms to meet the demands of massive sensor-based data analytics.

The analysis of large datasets presents several challenges, with the most important being the need for scalable and efficient computational architectures and algorithms capable of handling the volume, variety, and velocity of the data. Conventional data processing methods often fail to address these needs, as we discuss in Section 2.1. Furthermore, the complexity and heterogeneity of the data requirements, as well as the need for secure and privacy-preserving data handling, highlight the need for advanced analytic techniques and algorithms that can extract meaningful insights and patterns from vast and disparate data sources. The

intersection of data analytics with disciplines such as ML, data mining, and statistical analysis underscores the multidisciplinary approach required to address these challenges.

Massive sensor-based data analytics involves four key aspects: data collection, preprocessing, visualization, and analysis. Data collection encompasses the acquisition of data from a variety of sources, including sensors, databases, and web services, necessitating strategies to ensure data quality despite issues like missing data. Preprocessing follows, where data is cleaned and standardized to mitigate outliers and noise, which is critical for ensuring data quality and enhancing performance of subsequent analysis. Outlier cleaning and artifact rejection are essential preprocessing steps in healthcare applications, particularly when dealing with physiological signals such as electrocardiograms (ECGs), electroencephalograms (EEGs), or other biosignals. Visualization techniques, such as line and scatter plots, alongside advanced methods like autocorrelation plots, play a crucial role in identifying patterns and trends within the data. It aids in the interpretability and decision-making processes, which is particularly important in healthcare. These techniques help to communicate findings to healthcare professionals and patients, enabling informed decision-making.

The application of statistical and machine learning techniques is central to massive sensor-derived data analysis. These techniques enable the forecasting of future trends based on historical data, as well as pattern detection, feature interpretation, and comprehension of data structure. This deeper understanding of data allows for the extraction of actionable insights across various domains. In this work, we concentrate on exploring "shape generalization" for large datasets, and "similarity search" within the context of time-series data, positioning time-series analysis as a specialized instance of extensive sensor-oriented data processing. The field of time-series analysis has experienced significant growth due to its applicability across various domains where data is inherently temporal. Our focus on "shape generalization" and "similarity search" aims to address the inherent challenges posed by large datasets, thereby improving the efficiency and accuracy of time-series analysis.

"Shape generalization," in the context of extensive sensor-based data analysis, emerges as a critical tool in navigating the complexities of large datasets, including but not limited to the development of digital terrain models (DTMs). By distilling complex shapes to their most fundamental essence, shape generalization maintains data integrity and accuracy while significantly reducing computational demands. This simplification process is instrumental in handling the high volume and complexity that characterize massive datasets, thus facilitating more manageable and interpretable data representations. The adoption of

shape generalization is particularly salient in scenarios requiring precise topographical representation, such as DTMs, where the balance between data simplification and the preservation of essential terrain features is crucial. In the specific context of our work for obtaining an initial surface of reference for the construction of DTMs, shape generalization is instrumental in refining the terrain data, enabling the effective removal of extraneous details that do not contribute to the overall ground topography, while maintaining critical features such as elevation contours and terrain irregularities. For datasets devoid of topological details, such as Global Positioning System (GPS) trajectories or seismic recordings, line generalization methods like the Douglas-Peucker, Visvalingam-Whyatt, and Ramer-Douglas-Peucker algorithms offer effective solutions for simplification. However, when working with data that includes topological information, it is necessary to use alternative strategies. Techniques such as morphological filters, segmentation and clustering algorithms, and progressive densification, either individually or in combination, provide robust pathways for executing the generalization process. The use of advanced generalization techniques, adapted to the unique characteristics of terrain data, allows for the optimization of data processing and analysis, aligning with the challenges outlined in the preceding discussion on massive sensor-based data analytics in Section 2.1. This tailored approach ensures that the resulting DTMs accurately represent the terrain with reduced computational resources, thereby underscoring the indispensable role of shape generalization in enhancing the efficiency and effectiveness of algorithms designed for massive datasets. This synergy between shape generalization and computational performance improvements reinforces the arguments presented in Section 2.1 and Section 2.4, highlighting the pivotal role of suitable generalization techniques is crucial to tackle the challenges posed by large datasets and to advance analytical capabilities within the field.

The discovery of similarities, referred to as *motifs*, and anomalies, known as *discords*, in time-series data constitute a fundamental task in time-series analysis. The process of motif discovery involves the detection of recurring patterns within a time-series, without any pre-existing knowledge about their shape or location. In contrast, discord discovery aims to identify the most dissimilar patterns. Several techniques are available for similarity search, and their suitability depends on the nature of the data, the specific requirements of the application, and the constrains of the hardware architecture. Motif discovery is a versatile process, applicable to a wide array of data types, encompassing both one-dimensional and multidimensional, as well as univariate and multivariate datasets. Its versatility extends to analyzing both spatial and temporal data. The objective is to find the most similar pattern, known as the Nearest Neighbor motif (NN motif), or the

most frequent pattern ($k$-frequent motif). This approach tries to find either an exact or an approximate solution, tailoring the approach to the specificity and complexity of the dataset at hand. The similarity between time series can be determined using various distance measures, including Euclidean Distance (ED), Dynamic Time Warping (DTW), and Longest Common Subsequence (LCSS), among others. The selection of a distance measure is crucial as it has a significant impact on the analysis results and involves a trade-off between accuracy and computational cost. To enhance the search process's efficiency and scalability, the application of pruning strategies is essential, effectively narrowing the search space. Recent advancements in the healthcare sector have showcased the utility of pattern matching techniques to enhance the precision and efficiency of epileptic seizure detection through EEG recordings [104, 117, 137]. This evolution evidences the critical role of methodical selection and application of distance measures in refining analytical processes and outcomes. From a software architecture standpoint, the adoption of parallel and distributed computing approaches significantly augments the performance of these analytical techniques. Utilizing multi-threading, SIMD, and various acceleration hardware like GPUs and FPGAs facilitates the acceleration of the search process.

### 2.5.1. Algorithm Implementation

The experiments described in Chapters 3 and 4 employed a diverse set of programming languages and frameworks, chosen based on the development stage and target hardware architecture. In Chapter 3, we initiated our work by porting the original R implementation [16] to C/C++. This transition addressed memory constraints encountered in the R code when processing large LiDAR point clouds. Subsequently, we optimized the C++ code for various hardware accelerators using SYCL, CUDA, and oneTBB frameworks. For the study presented in Chapter 4, we developed the algorithm from scratch using Python. This ground-up approach allowed us to tailor the implementation precisely to our research needs, focusing initially on algorithmic correctness and functionality rather than performance optimization. As the development progressed and the algorithm matured, we identified computational bottlenecks in our custom code. To address these performance issues, we selectively optimized critical sections using SYCL and oneTBB frameworks. This strategy of building from scratch followed by targeted optimization enabled us to maintain full control over the algorithm's design. The combination of a custom-built foundation with strategic acceleration demonstrates our commitment to both algorithmic innovation and computational efficiency in the context of EEG data analysis.



While the LiDAR processing algorithms in Chapter 3 primarily relied on the core functionalities provided by SYCL, oneTBB, and CUDA for optimization, the EEG analysis in Chapter 4 necessitated additional high-level libraries to support its more complex data processing requirements. This distinction reflects the diverse nature of sensor data analytics and the need for specialized tools in certain domains. The following paragraph details the specific approach taken for the EEG algorithm development.

To extract data from `.edf` files, we utilized the WFDB Python package [135] [6]. This package, inspired by the original C-language WFDB software, provides a more accessible interface for researchers familiar with Python. WFDB enables the import of physiological signal files and associated annotations, facilitates signal plotting, and allows for the extraction of relevant features. For the pre-processing stage of EEG data in Chapter 4, we employed the PyRiemann Python package [9]. PyRiemann extends the scikit-learn API to provide advanced capabilities for processing and classifying multivariate data through Riemannian geometry of Symmetric Positive Definite (SPD) matrices. While designed with a focus on biosignal analysis (EEG, Magnetoencephalography (MEG), Electromyography (EMG)) for brain-computer interfaces, PyRiemann offers a versatile toolkit for multivariate data analysis. It specializes in estimating covariance matrices from multichannel time series and classifying them using the Riemannian geometry of SPD matrices. To optimize the searching process on the space of possible bounds for all features, we implemented a global optimization approach using the C++ library dlib [75]. Specifically, we utilized dlib's `global_function_search` function, which implements Lipschitz optimization [76]. This method aims to locate global optima while minimizing the number of function evaluations. To make the optimizer more user-friendly, a custom python wrapper [7] was developed to simplify the use of the optimizer. The wrapper is based on a general one [11] that was adapted to fit the specific needs of the algorithm. The wrapper handles the re-evaluation of the function to be optimized, the selection of the next set of categories (*next candidate*), and the update of the lower and upper bounds of the search space. The wrapper also enables the user to upload a previous set of evaluations, which is useful when the algorithm is stopped and needs to be restarted, or when the user wants to run the algorithm multiple times with different initial sets of categories.

[6] Our team contributed to the WFDB project by submitting a pull request to address specific data extraction issues, which can be reviewed at `https://github.com/MIT-LCP/wfdb-python/pull/435`

[7] Our research contributed to the "lipo" project by submitting a pull request to address specific issues, which can be reviewed at `https://github.com/jdb78/lipo/pull/193`.

This chapter highlights the importance and challenges of massive sensor-based data analytics. The domain encompasses various methods, tools, and applications that collect, process, and extract insights from large and fast-paced data streams. Navigating the world of massive sensor-based data analytics can be complex and challenging. To optimize performance, it is crucial to scale and fine-tune analytic processes while maintaining high standards for data quality, security, and privacy. Addressing these obstacles is essential to fully leveraging the potential of massive data analytics. Our research explores advanced computational models and techniques that promise to refine and improve data analytics practices. We highlight the challenges of analyzing vast amounts of sensor-based data in sections such as Section 2.1, which discusses the exponential growth of data, Section 2.4, which explores the role of software in handling massive data, and Section 2.3, which examines the hardware requirements and new architectures for processing large-scale datasets efficiently. By leveraging these advancements, we can gain more sophisticated insights and develop innovative solutions based on data, driving progress across various domains, from healthcare and finance to scientific research and beyond.



# 3 Accelerating Ground Point extraction from LIDAR data clouds

## 3.1. Motivation and summary

Light Detection and Ranging (LiDAR) is an active sensing technique capable of measuring distance between the sensor device and the target object [112] that has attracted the attention of the scientific community, as well as public administrations and private companies. LiDAR has become an established method for collecting accurate information of the Earth's landscape. The record of elevation data and the ability to penetrate through the canopy are some of the advantages over traditional acquisition methods like aerial imagery. The required equipment consists of an Airborne Laser Scanning (ALS), an Inertial Measurement Unit (IMU) and a Global Positioning System (GPS) receiver. These devices are installed in an aircraft that flies over the ground surface obtaining a georeferenced point cloud. On land a GPS system is synchronized with the GPS receiver on the aircraft. For each point, a set of features are recorded: $(x, y)$ coordinates, elevation $(z)$, intensity, pulse id, number of pulses, and scan angle. Photogrammetry is another sensing technique, which obtains point clouds from aerial images. The density of the point clouds is usually higher, because it can be configured by software, but the accuracy is worse, particularly in flat areas and wooded regions, due to the impossibility of penetrating the tree canopy and the need to establish ground control points, which are essential for providing the georeferencing

parameters and suppressing undesirable error propagation [36, 112]. Due to the huge size of the collected information with these techniques, and the ubiquity of heterogeneous parallel computing systems nowadays, the optimization and parallelization of the algorithms that process LiDAR point clouds are paramount in order to achieve efficient solutions for these architectures.

Computing the Digital Terrain Model (DTM) from ALS point clouds is one of the first steps to automatically obtain relevant information from the surface that has been scanned using LiDAR technology, and DTMs are becoming standard products available from national geoportals and private companies. The first step for DTM computation is the ground point extraction from the point cloud, i.e. the identification of the points in the cloud that correspond to the ground surface. After that, an interpolation step to derive the DTM is needed [112]. Our research focuses on a recent alternative in the process of ground point extraction, the Overlap Window Method (OWM) [16]. This algorithm was initially implemented in R, which resulted in that only small point clouds could be analyzed (less than 50k points in [16]). In order to cope with larger datasets, we have ported the code to C++ and optimized it to make the most of the current heterogeneous parallel architectures, featuring multicore Central Processing Units (CPUs) and Graphics Processing Units (GPUs). We believe our optimizations are also applicable to other algorithms for processing LiDAR data point clouds because the data structures to represent this type of point clouds and their traversal are similar to the ones implemented in our approach.

oneAPI [59] is an advanced framework that significantly streamlines the process of programming heterogeneous parallel architectures. It accomplishes this by offering a comprehensive suite of libraries, profiling tools, and analysis instruments, as previously introduced in Section 2.4. SYCL [121], a key component of oneAPI, is a royalty-free, cross-platform abstraction layer. This innovative technology allows software developers to compose a single codebase that can be compiled for a diverse array of heterogeneous parallel processors, embodying the principle of "write once, run everywhere". As discussed in Section 2.4.1, SYCL stands out as one of the most promising unified programming frameworks for heterogeneous computing, offering a potent combination of flexibility, performance, and portability. Currently, oneAPI's SYCL compiler is able to target multicore CPUs, GPUs and FPGAs. SYCL 2020 is the latest version of the standard and provides support for *Unified Shared Memory* (USM), ordered queues, reductions, hierarchical parallelism, subgroups (on CPU and GPU implementations), and data flow pipes (for FPGAs). The main benefit of using SYCL over other languages for heterogeneous programming, such as OpenCL or CUDA, is the portability of a single source code able to target multiple devices, which trans-

lates into cleaner and more easy to maintain code. Although oneAPI's SYCL compiler was initially devised to target Intel CPUs, GPUs and FPGAs, as an open source project, new backends have been added to also target architectures from other vendors. Among them, NVIDIA discrete GPUs (dGPUs) can be exploited, either by instructing the SYCL compiler to generate code for the dGPU or by embedding CUDA code inside SYCL code. Our research evaluates these programming models and assesses their performance impact on a platform based on commodity CPU and GPU devices.

Summarizing, this chapter covers the following contributions:

- We present several optimized parallel implementations of the OWM algorithm in C++, SYCL and CUDA, which are able to process point clouds of several million points on commodity devices (CPUs and GPUs) in less than a second.
- We discuss the tradeoffs between different data structures to store the LiDAR points (binary tree, quadtree and octree) as well as different techniques inherited from *branch-and-bound* and *ray tracing* algorithms to optimize the construction and traversal of the tree. Our findings are of interest to other applications that process LiDAR data point clouds.
- We contribute with parallel optimizations for CPU and GPU towards reducing memory bandwidth requirements which is key for the data structure construction and traversal in LiDAR memory bound problems.
- We compare SYCL, SYCL+CUDA and pure CUDA programming models in terms of performance and portability on our target platform, and provide the source code and results of all our implementations[1].

The rest of this chapter is organized as follows. Section 3.2 briefly introduces the OWM problem, suitable data structures, the testbed and the baseline performance that we strive to beat. Section 3.3 and Section 3.4 describe CPU and GPU oriented optimizations, respectively. The experimental evaluation is presented in Section 3.5 to finally conclude in Section 3.6 summarizing the main take-home messages.

## 3.2. Background

As commented before, the DTM is one of the most important products to be obtained from 3D point clouds from LiDAR and photogrametry. A DTM is a continuous function that maps from 2D planimetric position to terrain elevation

[1]Github repository: `https://github.com/...` (available upon acceptance).

$z = f(x, y)$ [92]. This function is stored digitally, together with a method on how to evaluate it from the stored geometrical and topological entities. DTM refers to the bare earth surface, without man-made or natural objects such as buildings and trees. Although DTM generation methods vary, most of them share several main steps (though conducted in different orders): data pre-processing, ground point filtering (the ground point extraction from the point cloud) and interpolation to derive the DTM. In [18] the general principles of DTM generation are introduced and diverse mainstream DTM generation methods are reviewed.

Although several ground filtering methods have been proposed based on different techniques [92], it is difficult to find one particular method that satisfies the requirements of every type of terrain. The OWM proposed in [16] is one of the most accurate filters in different scenarios that has been tested with the reference samples from the International Society of Photogrammetry and Remote Sensing (ISPRS), as is shown in the analysis performed in the mentioned work.

However, the main problem of this algorithm is its low computational performance, since it was initially programmed in R. In a previous work, the code was ported to C, demonstrating a good performance of the algorithm even without parallelization. It was also parallelized using OpenMP, achieving a considerable improvement in processing time, but high execution times (¿25 sec.) were still obtained when using dense clouds with a large number of points (¿20 Mpts and ¿7 pts/m$^2$). We propose to conduct a complete study of the algorithm, characterizing the bottlenecks and seeking solutions to overcome them, as well as leveraging current heterogeneous programming models like CUDA and SYCL for exploiting GPU and shared memory architectures.

Some works about the implementation in GPU of algorithms processing LiDAR data point clouds can be found in the literature. In [55] the Semi-Global Filtering (SGF) to classify points as either ground or non-ground is proposed. The authors claim that the SGF algorithm offers high classification accuracy and also it was parallelized using a GPU processing approximately 3 million points per second. In [54] a scan-line-based algorithm to detect local lowest points first and treat them as the seeds to grow into ground segments in presented. This algorithm was implemented in GPUs by processing each scan line independently. The test results show that the proposed algorithm is fast and effective in urban areas, but does not work well in mountainous areas. An efficient method for ground filtering of airborne LiDAR data based on scan-line processing is proposed in [108]. The algorithm was ported into a low-cost development board to demonstrate its feasibility to run in embedded systems, where throughput was improved by using programmable logic hardware acceleration. Analysis shows that real-time filtering is possible in a high-end board prototype, as it can pro-

cess the amount of points per second that current lightweight scanners acquire with low-energy consumption.

Related to obtaining the DTM, but using satellite instead of LiDAR, a procedure for building a DTM from the satellite stereo images is proposed in [34]. The procedure generates a DTM that may be less accurate than the one achieved with the use of the ENVI software, but it offers a significantly shorter processing time. In [129] an algorithm for the generation of Digital Surface Models (DSM) from satellite images was implemented on a GPU, increasing performance a 900% while having a negligible decrease in accuracy.

Our proposal is to optimize the OWM function, which is the most computationally intensive part of a reliable DTM algorithm proposed in [16], and to port it to GPU. More precisely, we present data structure and memory oriented optimizations that can be applied to other algorithms that process LiDAR point clouds. The resulting CPU and GPU optimized versions can process up to 68 and 285 million LiDAR points per second, respectively, which to the best of our knowledge is a performance not seen before in the context of robust ground surface detection algorithms.

### 3.2.1. The algorithm: OWM

The OWM function is part of the Hybrid Overlap Filter (HyOF) application [16], a hybrid algorithm for obtaining the DTM consisting of three processing blocks that are executed sequentially, from which the OWM is by far the most computationally intensive (consuming more than 80% of the total execution time). For this reason, in this chapter we tackle the problem of optimizing the OWM function, but please refer to [16] for more information about the other processing blocks.

As commented in Section 3.1, the OWM function aims at defining an initial reference surface from which to obtain the DTM, starting from a LiDAR point cloud. This is achieved by adjusting some parameters so that, in the successive iterations or steps, the points of the cloud that do not belong to the ground ($P_{ng}$; *non-ground point*; points that may belong to a rooftop, the top of a tree, etc.) are filtered out and the seed-points ($P_S$; *seed-points*, the key points that represent the ground surface preserving all the details of the relief) are collected, and from them the initial reference surface ($\varphi_{t=0}$) can be generated.

OWM is defined by 5 arguments: input data ($P$; *Point Cloud*); Bounding box ($x_{min}$, $y_{min}$, $x_{max}$, $y_{max}$) defined by the maximum and minimum values of the $(x, y)$ coordinates of the points in $P$; sliding window size ($\omega$) for the selection of

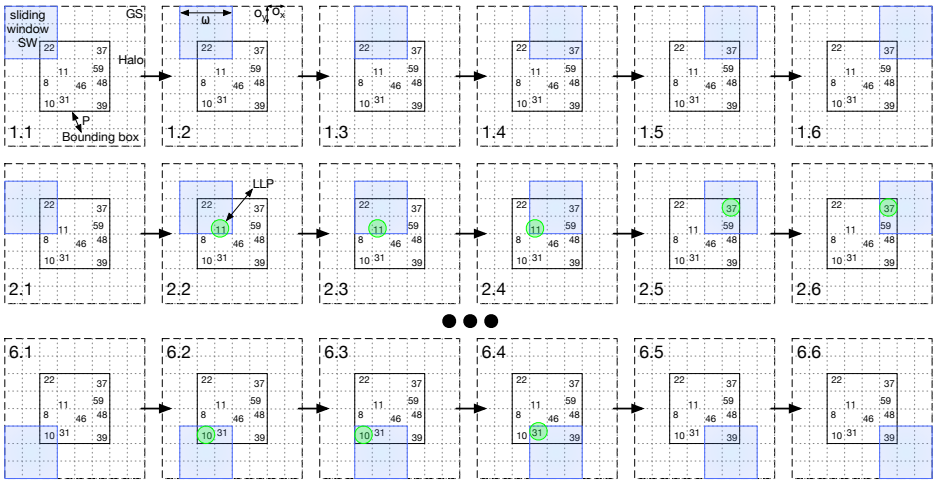



Figure 3.1: OWM process to select LLPs (identified by a green circle). The point cloud, $P$, is depicted by the numbers (the $z$ coordinate of each point). Only 18 sweeping steps out of the required 36 are depicted.

the Local Lowest Point (LLP); and horizontal ($O_x$) and vertical ($O_y$) overlaps between two consecutive windows. Both overlaps, $O_x$ and $O_y$ are expressed as a fraction of the window size so that we can define the overlap distances, $o_x = (1 - O_x) \cdot \omega$ and $o_y = (1 - O_y) \cdot \omega$.

Figure 3.1 illustrates the OWM function for a toy example. We assume a point cloud $P$ comprising just 10 points with coordinates $(x, y, z)$ confined in a bounding box. In the figure we depict each point by the value of its $z$ coordinate (point altitude) at approximated $(x, y)$ coordinates. To avoid an edge effect in the LLP selection, the area to be scanned is enlarged by $\omega \cdot O_x$ meters on both sides of the $X$ dimension and $\omega \cdot O_y$ meters on both sides of the $Y$ dimension. As a result, an empty halo surrounds the bounding box and the extended area is referred to as the "Geographic scope" (GS). In our example, $O_x = O_y = 2/3$ so that $o_x = o_y = \omega/3$ and GS becomes a grid of $o_x \times o_y$ boxes. The bounding box is a $4 \cdot o_x \times 4 \cdot o_y$ square, and GS dimensions are $8 \cdot o_x \times 8 \cdot o_y$.

With that, the OWM process goes by sweeping the whole geographic scope, GS, in a number of steps in which the Sliding Window (SW) scrolls in steps of $o_x$

or $o_y$, so the overlapping area of the SW when moving in the x direction is $O_x \cdot w^2$. In Figure 3.1 we show only the first 12 and the last 6 steps out of the total 36 steps that are required to cover the whole GS area. At each step, the point with the lowest elevation inside the SW is selected as an LLP (Local Lowest Point, and depicted inside a green circle in the figure) if the number of points under the SW is larger than $minNumPoints$. In this example we set the threshold $minNumPoints = 1$. Note that each $o_x \times o_y$ cell falls inside the sliding windows a number of times (9 times in our example and in general $1/(1-O_x) \times 1/(1-O_y)$ times). This means that a local minimum can be selected as LLP more than once, and up to 9 times in our running example. Clearly, the more times a point is selected as LLP, the higher the probability that it belongs to the ground ($P_g$; *ground point*). Although not explicitly shown in the figure, it is an easy exercise to validate that the point with $z = 11$ is selected 5 times, whereas the points with $z = 22$ and $z = 46$ are never selected (never surrounded by a large enough number of larger values). Therefore, the point with $z = 11$ is more likely to be a ground point than the point with $z = 46$. After 36 steps, points with $z$ equal to 11, 37, 8, 39, 31 and 10 are found as representative minimums or LLPs, 5, 4, 9, 4, 2 and 2 times, respectively. On the other hand, points with $z$ equal to 22, 59, 46 and 48 are never found as LLP inside a dense enough sliding window. In [16] it is demonstrated that the best heuristic consists in considering as a seed-point of the ground surface, $\varphi_{t=0}$, only the minimums that have been found more than once, since they are the most "reliable" minimums. Therefore, in our example the set of ground seed-points is $P_S = \{11, 37, 8, 39, 31, 10\}$ (using their $z$ value as their ids).

In our implementation we consider two phases in the OWM function: a data structure construction phase and a OWM traversal phase. The first phase is the input of the second one, which represents the main functionality of OWM. Algorithm 1 shows the pseudocode of the OWM traversal phase. The function receives as input the point cloud $P$, the sliding window size $\omega$, and the horizontal and vertical overlaps $O_x$ and $O_y$. The code computes first the number of steps required to cover the whole point cloud area (in negligible execution time) and then iterates over all the steps. At each step, the center of the sliding window is computed (line 5), and the points that fall inside this window are stored in the set *coveredPoints* (line 6). If the number of covered points is large enough (greater than $minNumPoints$) the id of the point with the lowest $z$ value in this set is found and stored in the array $LLPsFound$. The value of $minNumPoints$ is computed as $P.density \times \omega^2/2$ and therefore an LLP is the minimum z-value of a sufficiently dense window. After the loop nesting (line 14), the set of seed-points of the ground surface, $P_S$, is computed from the array $LLPsFound$ by

**Algorithm 1:** OWM traversal phase

```
   Input: P, ω, O_x, O_y      // Point cloud, sliding window size, horizontal and
          vertical overlaps
   Output: P_S                // Set of seed-points of the ground surface
 1 nRows, nCols = ComputeSteps(P, ω, O_x, O_y)
 2 countLPP = 0
 3 for i = 0, nRows do
 4 |   for j = 0, nCols do
 5 |   |   SW_Center = ComputeSlidingWindowCenter(P, ω, O_x, O_y, i, j)
 6 |   |   coveredPoints = search(P, ω, SW_Center)
 7 |   |   if coveredPoints.size() > minNumPoints then
 8 |   |   |   LPPid = Min(coveredPoints)
 9 |   |   |   LPPsFound[countLPP] = LPPid
10 |   |   |   countLPP++
11 |   |   end
12 |   end
13 end
14 P_S = ComputeSeedPoints(LLPsFound)   // Less than 0.07% of total execution
     time
15 return P_S = P_S ∪ FillEmptyCells(P, B, GroundSeedPoints)   // Less than 0.57%
     of total execution time
```

identifying the LLPs selected more than once. This function takes less than 0.07% of the total execution time for large point clouds since it only needs to sort the $LLPsFound$ array and iterate it by comparing consecutive entries.

In high-slope or very sparse zones, it is unlikely that the same point will be identified as an LLP more than once, and the resulting seed-point cloud will have some zones without points. In order to minimize the lack of detail in these zones, a grid of square cells of size $B \times B$ is overlapped to the geographical scope (see [16]). For those cells without points, the lowest point is selected from the original point cloud and is added directly to the seed-point cloud achieving the final OWM result, as can be seen in line 15 of Algorithm 1. In our experiments (see later), this phase takes less than 0.57% of the total execution time. Other steps to construct the final ground surface out of the seed-point cloud are described in [16], but they are not considered here since they represent a negligible amount of execution time and do not require any optimization strategy.

The loop nesting starting at line 3 of Algorithm 1 is the most time-consuming part of the OWM traversal phase taking more than 99.4% of the total execution time. A first optimization consists in parallelizing the outer loop using OpenMP adding a parallel directive to the "i" loop, with the `schedule(dynamic)` clause (to account for load unbalance when traversing the point cloud data structure)

and a critical section protecting lines 9 and 10. This results in a 2-6x speedup in our 8-core platform. Other OpenMP strategies could be implemented in order to improve the parallel performance. For instance, we used the "collapse" clause to linearize the loop nesting and explored different scheduling policies: static, guided, and dynamic. Moreover, for dynamic we looked at different scheduler granularities (chunk sizes), from 1 to 14. Using "collapse" and the best chunk size we got OWM traversals that were ≈24% faster for dense clouds but ≈17% slower for sparse ones. With these findings, we will consider the OMP version without "collapse" and dynamic scheduler with chunk size of 1 as our baseline because it is representative enough of a reasonable OpenMP implementation that an average developer would consider acceptable. The corresponding traversal parallel times are shown in Section 3.2.5.

### 3.2.2. The data structure: trees

The point cloud, either stored in a file or arriving as a stream from a LiDAR device, is unsorted, and there is not a clear sorting criterion. However, as described in the previous section, at each step of the OWM algorithm we have to find the point with the minimum value of $z$ from the set of points for which the two other coordinates, $(x, y)$, fall into the corresponding sliding window. Therefore, to optimize each OWM step it is required to efficiently access and identify the points inside a sliding window. This is a memory-bound problem, so data locality should be exploited, thus it is a must to keep close in the data structure the points that are geographically close.

To solve this issue, hierarchical data structures are widely used in fields like graphics, image processing, geographical information systems, and robotics among others. These hierarchical data structures are based on the divide and conquer principle and on keeping related data close in the data structure. At the top of the data structure we have a handle that gives access to all the elements stored. Using recursive decomposition, lower levels of the data structure store fewer elements, which tend to be more closely related if they share a node than if they are in different nodes. With this idea, it is possible to traverse the data structure, from top to bottom, to different regions of related data, and back bottom-up, to gather information obtained at lower levels. A wide range of hierarchical data structures has been studied in the field of spatial data representation. Among these, tree-like data structures (binary, quadtree and the $\mathbb{R}^3$ alternative *octree*) have proven their suitability for our problem domain. In a space storing 3D regions objects and point clouds are usually recursively partitioned in 8 sub-cubes per cube, which naturally translate into octrees.

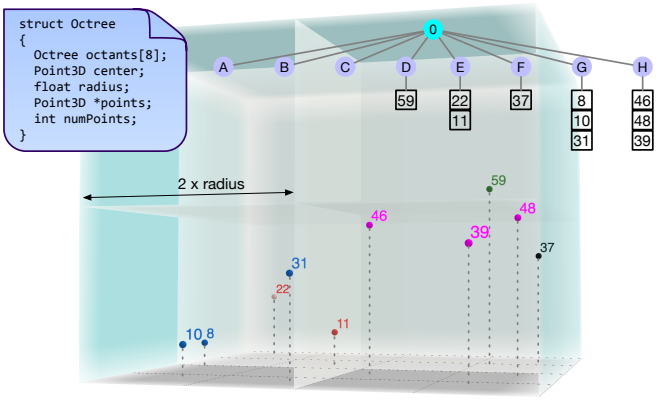



Figure 3.2: Octree example and C declaration of the data structure. The color of each point in the cloud depends on the occupied octant.

Figure 3.2 shows a possible octree for our running example and the corresponding C data structure declaration. The root node, 0, is the handle to access the whole point cloud. The bounding box, $[(0,0,0),(100,100,100)]$, is bipartitioned along the three axis in eight equal sub-cubes or octants that are represented by 8 nodes ($A$ to $H$). In the data structure, each octant could have been identified by the minimum and maximum coordinates of the corresponding bounding box $[(x_{min}, y_{min}, z_{min}), (x_{max}, y_{max}, z_{max})]$, but each node occupies less memory storing only four values: center $(x_c, y_c, z_c)$ and radius (that is actually half the side length of the sub-cube). In this example, three octants are empty and the number of points in the other ones varies between 1 and 3. Dense enough octants can be further subdivided recursively, but in our toy example this is not recommended if we do not want to waste more memory in empty nodes.

In memory-bound problems like the one at hand, minimizing the memory footprint is key to reduce data movement and improve locality, so in the next section we will propose several data structure optimization alternatives targeted to minimize execution time in both CPU and GPU architectures.

### 3.2.3. The target architectures: CPU and GPU

CPU and GPU architectures, and their memory models, are quite different. Thus, to get the most out of each device, algorithms and data structures should be tailored accordingly. GPUs are accelerators that excel at exploiting massive data-

level parallelism but at the price of banning some operations that are available in general purpose processing units, like dynamic memory allocation and recursion, among others.

On a CPU, recursion and dynamic memory are usually leveraged to construct and traverse recursive data structures like trees (a node points to other nodes). On the other hand, the GPU works more efficiently with arrays, which means that tree-like data structures are usually implemented on GPUs using one or more arrays, and indices are used instead of raw pointers. Besides, total space required to store the tree has to be preallocated in advance in GPU memory.

Solving the OWM function implies constructing the tree out of the LiDAR point cloud, to later traverse it finding the LLPs that will serve to identify the ground points of the Digital Terrain Model. The next section describes the tree construction and OWM traversal phases for both the CPU and GPU, but first, we elaborate next on the suitable programming model able to target both CPU and GPU architectures.

### 3.2.4. The programming model: oneAPI vs CUDA

Traditionally, NVIDIA GPUs and the CUDA programming language have dominated the General-Purpose Graphics Processing Units (GPGPU) arena, especially for High Performance Computing (HPC) problems. CUDA has evolved for more than 20 years and provides high level APIs, which are used by several Domain Specific Languages (DSLs) such as Tensorflow, PyTorch, etc., but also offers low-level and highly optimized routines that make the most out of the NVIDIA architecture. However, CUDA is a proprietary language targeting only NVIDIA GPUs.

In the context of accelerating massive sensor-based analytics, a promising alternative emerges in the form of the oneAPI open industry standard, which we previously discussed in Section 2.4. This standard aims to provide a unified ecosystem of languages, libraries, and tools capable of targeting diverse architectures (CPU, GPU, FPGA, etc.) in an accessible and efficient manner. The Intel implementation of oneAPI includes the `icpx` compiler, built on LLVM, which can compile SYCL code—potentially enhanced with Intel extensions—into binaries compatible with Intel CPUs, GPUs, and FPGAs. Furthermore, SYCL code can be executed on NVIDIA GPUs when targeted as OpenCL devices, and a recent CUDA backend for SYCL even allows the compilation of SYCL code incorporating native CUDA kernels. In this chapter, we leverage these capabilities to optimize the OWM algorithm for various architectures, including CPU multi-

cores, integrated GPUs, and NVIDIA discrete GPUs (including native CUDA implementations). We then conduct a comprehensive analysis comparing both the portability and performance of these optimizations, with the results presented in the experimental section. This approach aligns with our broader goal of developing efficient, portable solutions for processing large-scale sensor data across diverse computing platforms.

Furthermore, oneAPI incorporates the oneTBB library [131], which offers a sophisticated parallel programming model. This model is founded on task parallelism and employs a work-stealing scheduler, effectively addressing workload imbalances. As previously discussed in Section 2.4.3, oneTBB is built upon the C++17 standard and provides a suite of parallel templates. Of particular interest is the `parallel_for` template, which we will leverage to efficiently implement a parallel version of the OWM algorithm.

### 3.2.5. The testbed: HW, dataset and baseline

| | Number of Points [M] | Bounding Box [$m$] | Dim. [$m$] | Density [$pts/m^2$] | Tree C. seq. [$s$] | OWM seq. [$s$] | OWM par. (8ths) [$s$] |
|---|---|---|---|---|---|---|---|
| Alcoy | 20.4 | 714948<br>4286502<br>716361<br>4288406 | 1413<br>x<br>1904 | 7.57 | 4.74 | 22.55 | 3.73<br>(6.0x) |
| Arzua | 40.7 | 568000<br>569000<br>4752320<br>4753320 | 1000<br>x<br>1000 | 40.7 | 5.75 | 22.94 | 4.11<br>(5.6x) |
| Brion Forestal | 42.4 | 526964<br>4742610<br>527665<br>4743116 | 700<br>x<br>505 | 119.7 | 5.97 | 19.75 | 6.23<br>(3.2x) |
| Brion Urban | 48.0 | 526956<br>4742586<br>527686<br>4743124 | 730<br>x<br>538 | 122.11 | 6.76 | 21.30 | 7.33<br>(2.9x) |

Table 3.1: Details of the LiDAR point clouds used in the experiments.

Our test platform includes a $9^{\text{th}}$ generation Intel processor with Coffee Lake architecture. More precisely, the CPU is the Core i9-9900K with 8 cores at 3.60 GHz and 8x256 KB L2 cache plus 16 MB shared L3 cache and a 32 GB

DDR4 memory (37.7 GB/s BW[2]). This processor also features an integrated GPU Intel UHD Graphics 630 @ 1200 MHz (24 EUs). Additionally, the board includes a NVIDIA discrete GPU GeForce RTX 2060 @ 1755 Mhz (30 SMs) and 6 GB of VRAM GDDR6 (131.2 GB/s BW[3]). The operating system is Ubuntu 20.04.2 LTS, kernel 5.11.0 and we use the GCC 9.3.0 compiler and `icpx -fsycl` (previously known as `dpcpp`) compiler from oneAPI version 2024.0. For now on we will refer to this platform with the code name `Coffee`. We compile with optimization flag -O3 and reported execution times are the average of 5 runs in which the server is not running any other application.

All of the experiments in this chapter will be done using the `Coffee` platform we just introduced. Whenever we want to support our conclusions or deepen the explanation, we will rely on two other platforms:

- `Alder`: 12$^{\text{th}}$ generation Intel processor with Alder Lake architecture, already introduced in Section 2.3.2.
- `Bombay`: Intel CPU Max Series processor with Sapphire Rapids HBM architecture. The CPU is the Xeon Max 9468, featuring 48 cores operating at 2.10 GHz, Thermal Design Power of 350 W, 48x4 MB of L2 cache, 210 MB of shared L3 cache and 1 TB DDR5 memory. This processor counts with 64 GB of HBM memory in "Cache Mode", where the HBM is invisible for both OS and application and is used as a L4 cache for the DDR5 memory, which avoids the need to explicitly manage the data movement between the two memory types.

The dataset comprises 4 LiDAR point clouds[4] from different regions with the following names: Alcoy, Arzua, Brion Foresta and Brion Urban. Table 3.1 summarizes the main characteristics of these point clouds, including the number of points, bounding box ($x_{min}$, $y_{min}$, $x_{max}$, $y_{max}$), spatial dimensions and point density (pts/m$^2$). It should be noted that this is the average density, although in the Brion clouds (obtained by photogrammetry) there are areas with very high density and others with very low one.

In our experiments we have fixed $O_x = O_y = 4/5$ and $\omega = 10$ m. This results in $o_x = o_y = 2$ m for the sliding window displacement. This means that the overlapping area is 80 m$^2$ and also that the same LLP can be selected up to 25 times. Under these conditions, Table 3.1 also shows the execution time of the C sequential code and OpenMP version that was described in Section 3.2.1

[2]Reported by the Intel Advisor profiler
[3]Reported by the NVIDIA Nsight profiler
[4]Provided by Babcock International, later known as Avincis, before being acquired by Ancala Partners in 2023: `https://www.avincis.com/`

specifying both the tree construction sequential time, OWM traversal sequential and OWM traversal parallel times with 8 threads. Note that the tree construction time grows with the number of points in the cloud, however, the OWM traversal time depends more on the bounding box and therefore the number of steps. For example, Alcoy requires $714 \times 959$ steps and although it has lower density than Brion urban, the former requires more OWM traversal time than the later larger cloud that only requires $372 \times 276$ steps. We also see that the parallel speedup is particularly poor for the larger clouds. As we will see later, that is due to the memory-bound nature of this algorithm. We would like to remark that this baseline implementation is already a step forward in comparison with the original R implementation of the OWM function. This original code was evaluated in [16] with small point clouds (less than 52,000 points) and out of the clouds of Table 3.1 only Alcoy can be processed in R, taking 15 minutes (the other larger clouds run out of memory).

We will consider the times reported in Table 3.1 as the baseline that we tackle to outperform with the optimizations described in the next section, studying their impact on the OWM traversal phase -that is slower-, but also on the tree construction phase that becomes the bottleneck once the OWM phase is optimized. Our optimizations are organized in four categories and are adapted to target the specific features of each device on our platform: a CPU multicore and a GPU.

## 3.3. Optimizations targeting the CPU

In this section, we present different optimizations aimed at reducing the CPU execution time by devising better data structures, improved parallelization techniques and memory access reductions.

### 3.3.1. Optimization 1: 2D-space bipartition

In airborne LiDAR, since the aircraft is usually hundreds of meters above the area of study, the point clouds are, so called, 2.5D because the range of values in the z-axis is small in comparison with the range in the other two dimensions. Consequently, using a 3D bipartion of the space results in many empty octants (as we saw in Section 3.2.2), which translates in a lot of memory wasted for large point clouds. To alleviate this problem, we propose to project the points into the X-Y plane (using the $(x, y)$ coordinates of the points) and to partition this 2D space instead. Now, the bounding box is recursively partitioned into quadrants and therefore a quadtree is the natural data structure to store the point cloud.



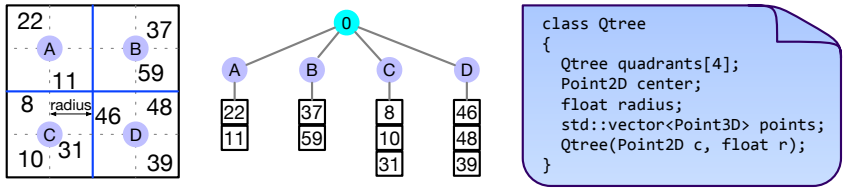



Figure 3.3: Quadtree example and C++ class alternative for the data structure.

Figure 3.3 shows a possible quadtree for the same running example of Figure 3.2. Now we use C++ so we also show the corresponding C++ class declaration (including the constructor declaration in the last line). The root node, 0, gives us access to the whole point cloud, but now the 2D bounding box is partitioned into four equal quadrants that are represented by 4 internal nodes ($A$ to $D$). Again, instead of identifying each quadrant by its bounding box coordinates, $[(x_{min}, y_{min}), (x_{max}, y_{max})]$, it is cheaper to store only the three values needed for the center $(x_c, y_c)$ and radius. In this example, there is no point in further subdividing each of these nodes into smaller ones because seven of them would be empty. The alternative is to consider nodes $A$ to $D$ as leaf nodes that include a vector of the 3D points that fall in each quadrant.

### 3.3.2. Optimization 2: CPU parallelization

Regarding the parallel programming model for the CPU, we moved from OpenMP (where a `parallel for` was used in the baseline implementation) to Threading Building Blocks (oneTBB) [131]. The main advantage of TBB relies on the `parallel_for` template that we have used to parallelize the main loop of the OWM traversal phase. The mentioned template is based on a work-stealing task scheduler that naturally deals with load unbalance and locality. It is, if a worker thread finishes its tasks earlier, it can steal tasks from other busier worker threads (usually the coldest in the cache used by the stolen task). This is a key feature in our problem where the OWM steps are processed in parallel, but each step has an unknown amount of workload depending on the corresponding slide window coordinates and on the density of the point cloud at this region of the 2D space. In addition to the OWM traversal phase parallelization, we also explore how to parallelize the tree construction phase.

#### 3.3.2.1. OWM traversal phase parallelization

With respect to the baseline OWM traversal phase described in Algorithm 1 of Section 3.2.1, we have identified three main optimization strategies that are sketched in the pseudocode of Algorithm 2.

**Algorithm 2:** Optimized OWM traversal phase

```
   Input: P, ω, O_x, O_y      // Point cloud, sliding window size, horizontal and
          vertical overlaps
   Output: GroundSeedPoints   // Set of seed-points of the ground surface
1  NumSteps = ComputeNumSteps(P, ω, O_x, O_y)
   /* For loop parallelized with tbb::parallel_for                          */
2  for step = 1, NumSteps do
3  |  SW_Center = ComputeSlidingWindowCenter(P, ω, O_x, O_y, step)
4  |  LLPid, numPts = searchMin(SW_Center, ω, PointCloud)
5  |  if numPts > minNumPoints then
6  |  |  LLPsFound[step] = LLPid
7  |  end
8  end
9  GroundSeedPoints = ComputeSeedPoints(LLPsFound)
10 return GroundSeedPoints = FillEmptyCells(P, B, GroundSeedPoints)
```

The first one consists in fusing the two nested loops that were traversing the rows and columns of the 2D space of the point cloud into a single loop that traverses all the steps, as we see in line 2 in Algorithm 2. This is now the loop that we parallelize with the TBB `parallel_for` template. This loop fusion enlarges the loop trip count which in turn increases the parallel scheduling opportunities for the loop. As the second optimization strategy, we have eliminated the critical section (that was protecting lines 9 and 10 in Algorithm 1) because now each thread is writing in the `step` (loop counter) position of the $LLPsFound$ array instead of using the induction variable `countLPP`. Finally, instead of first gathering the points that fall inside the sliding window and then finding the minimum if the number of points is large enough, now we directly compute the minimum and count the number of points, `numPts`, at the same time (function `searchMin` in line 4). Only if `numPts` is larger than the threshold, we store the index of the minimum point in the $LLPsFound$ array (which is initialized with -1's so that we can identify the updated positions).

#### 3.3.2.2. Tree construction phase parallelization

When constructing the tree on the CPU, two critical issues have to be efficiently tackled: i) minimize the synchronization among the threads and ii) parti-

tion the load carefully. Note that each LiDAR point initially stored in the input array has to be placed in the right tree leaf according to its ($x$, $y$) coordinates and that each branch depth may depend on the cloud density at the corresponding 2D region.

Our tree construction implementation consists of three stages as follows:

1. First stage (sequential): We sequentially construct the tree but only up to a given tree level, $lev$. For example, if $lev = 2$, only the root (level 0) and levels 1 and 2 are created, this is $4^0+4^1+4^2$ internal nodes. This is executed very fast, and it is not worth the overhead of creating and synchronizing the threads to parallelize it. This first stage uses the node radius as the tree splitting criterion.
2. Second stage (parallel): We traverse in parallel the array of LiDAR points. Each thread only visits a chunk of the input array and inserts the points in the internal nodes of the last level created in the previous stage. This is, these internal nodes are temporally used as leaf-nodes and they have a `tbb::concurrent_vector` data member in which the points can be stored. We rely on the TBB's `concurrent_vector` container because it allows for the concurrent insertion of elements in the vector. This is necessary because several threads may found in parallel that a point belongs to the same (temporary) leaf-node and the concurrent insertion has to be done in a thread-safe way. This is, this parallel stage can suffer from some parallel overhead due to potential collisions at the insertion of points.
3. Third stage (parallel): We traverse in parallel the (temporary) leaf-nodes (those at level $lev$) to finish up the tree construction. Basically each thread takes care of a temporary leaf-node and takes it as if it were a root node with a private vector of points to be inserted. That way each thread, in parallel, is able to construct the subtree (or treelet) hanging from the temporary leaf-node assigned to it, without any synchronization with the other threads. This final construction can be done considering different recursive cut-off criteria (to stop the recursive bipartition) that we explore in Section 3.3.4.

In essence, the first and second stages are implementing a pre-sorting phase in which the LiDAR points are classified according to the treelet to which they belong. This enables a fully parallel final stage that finish the construction of the tree. The caveat here is the proper selection of the level, $lev$, parameter. If $lev$ is too small, there will be less (temporary) leaf-nodes after the first stage which means more potential collisions, less parallelism available at the third stage, and larger temporary vector sizes. On the other hand, if $lev$ is too large, the first



stage will take longer, and it can create nodes associated to 2D regions without points or with very low density. In the experimental section we study the impact of *lev* in the tree construction time.

### 3.3.3. Optimization 3: reduce the number of accesses by memoization

In general, the smaller the depth and number of nodes of the tree, the smaller the memory footprint, but then the higher the number of accesses to check the points stored in the leaf-nodes. One strategy to reduce the memory bandwidth pressure consists in reducing this number of accesses. The function `searchMin()` (see line 4 of Algorithm 2) has to look for all the points inside the sliding window to find the minimum value. Due to the sliding window overlap there are points that are re-visited in different steps of the traversal. We can save some of these repeated accesses via memoization (storing values that have been computed before and reusing them when needed).

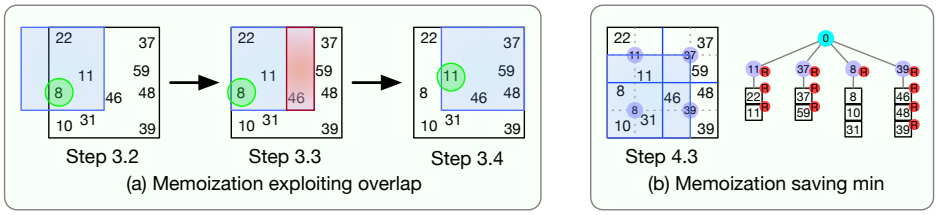



Figure 3.4: Two alternatives to exploit memoization in the OWM traversal phase.

In our problem we can keep the value of the minimum of a previous step and, if this minimum falls inside the sliding window of the next step, avoid searching in the area overlapped by both consecutive steps. For example, in Figure 3.4 (a) the minimum in step 3.2, 8, is still inside the sliding window in step 3.3. This means that we can just search for the minimum in the new region (the reddish area depicted in step 3.3) and keep the minimum of both regions (8 again, that is smaller than 46). However, this is not always the case as we can see in step 3.4, where the previous minimum, 8, is outside the new sliding window and the whole area has to be considered when finding the minimum. Note that the larger the overlap values ($O_x$, $O_y$), the more likely is that the previous minimum is inside the next sliding window and therefore that this strategy has an impact on the execution time. Pondering over also explaining that: the number of points under the SW in the case of step 3.3 is estimated since we are only visiting a fraction of the SW. This estimation is performed by calculating the point density

in the portion of the SW that is visited, then extrapolating to the entire SW. As a consequence, the selection of LLPs may differ slightly from the exact solution obtained without this optimization.

Another alternative consists in storing at each tree node the minimum value of all the elements hanging from that node. This minimum can be computed at tree construction or during the OWM traversal. With that, if the sliding window fully overlaps within the bounding box of a node, the corresponding subtree is not traversed because the node already provides the minimum of that subtree. Similarly, the number of points hanging from each node are also stored in that node, which avoids traversing the current subtree to gather this required information. The drawback of this approach is that the memory footprint of the tree is increased by the additional information stored in each node (minimum and number of points covered by each node).

This strategy is illustrated in Figure 3.4 (b) where we see the sliding window at the position corresponding with step 4.3 of our running example and the corresponding quadtree. Now the nodes of each quadrant (previously identified with letters $A$ to $D$) store the minimum of the points of that subregion in addition to the `points` vector. In the figure we identify with an "`R`" in a red circle the read accesses. Note that the points of node $C$ are not accessed because this quadrant is fully covered by the sliding window and the minimum of the subtree, 8, is already stored in node $C$. The vector of points of the other quadrants is completely traversed to check if each point falls inside the window and compute the minimum in such a case. This technique can be useful also for partially overlapped nodes. For example, node $A$ identified now with the minimum 11, is partially overlapped by the sliding window, but the minimum 11 is inside the overlapped region so we skip the minimum search. However, we still have to traverse the `points` vector to count the number of points that fall in this overlapped region.

These two approaches ((a) and (b)) are not exclusive, so we have put to work both of them during the minimum search at each step of the OWM algorithm. First, we identify the search region, which can be a fraction of the sliding window (as in Figure 3.4 (a) Step 3.3) or the whole sliding window (as in step 3.4). Then, we traverse the nodes of the tree from the root. If the region represented by a tree node is fully overlapped by the search region, we take the minimum and number of points stored in that node and do not navigate deeper in this tree branch. Otherwise, we visit the four children of the node and recursively do the same query. The impact of these approaches is analyzed in the experimental section.

### 3.3.4. Optimization 4: Tune the Granularity

Recursive data structures, as recursive functions, require a cut-off or base criterion (stop condition) to avoid infinite recursion. By default, tree data structures that store space related objects or points, stop the recursive partitioning of the space when the leaves of the tree store a single element/point. However, in our OWM case, having one point per leaf-node is too fine-grained, resulting in very deep trees with too many internal nodes (that need to be traversed) and the corresponding memory overhead. We propose two alternatives that help in this regard:

- **MinRadius**: a node of the tree is not further subdivided in quadrants if the resulting $radius$ of the leaves is smaller than $MinRadius$. In other words, all the tree nodes have $radius \geq MinRadius$. Note that this criterion does not consider the point cloud density so the sparsest regions of the cloud point have less populated leaf nodes which translates into load unbalance. Times reported in Table 3.1 were obtained with this technique and $MinRadius = 0.1$ m. In the same conditions and as an example, reducing $MinRadius$ to 0.01 m increases tree construction time up to 3x and OWM time between 2x to 6x depending on the point cloud size.
- **MaxNumber**: a node of the tree is not further subdivided in quadrants if the number of points contained in this node (`points.size()` in the data structure of Figure 3.3) is smaller than MaxNumber. This is, MaxNumber is the maximum number of points that can be stored in a leaf-node. If a new point has to be added to an already full leaf-node, that node turns into an internal-node, splitting the region in four quadrants (new leaf-nodes) and distributing the points (including the new one) among these children nodes. As a consequence, the depth of the branches depends on the density of the point cloud in the region represented by these branches but, at the same time, the load balance is better preserved for point clouds with significant differences in the densities of the subregions. On the other hand, converting a leaf-node into an internal node with four leaves and redistributing the points of the original leaf between the new leaves is a costly operation that it not always compensated by the extra load balance achieved later during traversal.

Figure 3.5 illustrates the differences in constructing the quadtree for these two covered alternatives. The blue dots represent the points of the cloud, which will be stored in the tree. As we can see in the tree with $MinRadius = o_x/2$, nodes $A$, $C$ and $D$ have to be split into quadrants with $radius = MinRadius$, resulting in leaf-node $d4$ storing the three points of the most dense region of the

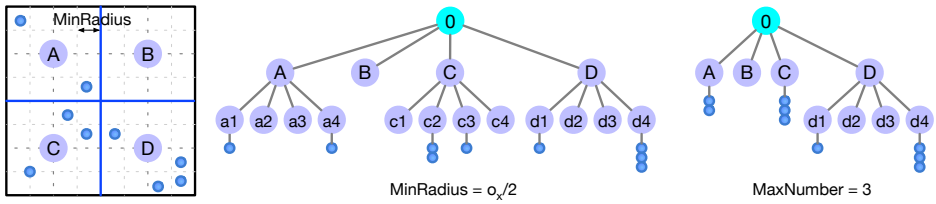



Figure 3.5: MinRadius vs. MaxNumber example.

cloud. Note that node $B$ does not have children because it is empty and that many leaf-nodes are also empty in sparse regions of the space. On the other hand, the tree with $MaxNumber = 3$ has a more balanced distribution of the points among the leaf-nodes and has fewer nodes and fewer empty nodes.

Either $MinRadius$ or $MaxNumber$ can be used to control the granularity of leaf-nodes and should be carefully tuned. Small values of $MinRadius$ or $MaxNumber$ result in a deeper tree with fine-grained leaf-nodes, more memory overhead and more time consumed at tree construction. On the other hand, large values lead to large `points` vectors in the leaves. These vectors will need to be fully traversed to read the $(x,y)$ coordinates and check if they fall inside the sliding window, and in such case, read the $z$ value to find the minimum.

Depending on the point cloud and target architecture (CPU or GPU) one criterion can be better than the other. In our experiments, we have found that $MinRadius$ is better for the CPU (the tree is constructed faster, and the load unbalance can be handled on the CPU with dynamic or task-stealing scheduling), whereas $MaxNumber$ is more suitable for the GPU (load unbalance has a deeper impact in this architecture and the tree is constructed differently so that there is no leaf-node to internal-node conversion overhead). In Section 3.5 we will show the results of the experiments with both alternatives.

## 3.4. Optimizations targeting the GPU

The GPU kernels for the tree construction and OWM traversal have been implemented in SYCL and CUDA. The SYCL code can be compiled for the CPU multicore device (S-CPU)[5], the integrated GPU (S-iGPU), or the discrete NVIDIA GPU (S-dGPU), as sketched in Figure 3.6. This code can be compiled

[5] "S" stands for SYCL.

using -DSCPU, -DSiGPU or -DSdGPU, and depending on the compilation flag the `queue` object will feed the CPU, the integrated GPU or the NVIDIA GPU, respectively. This feature greatly simplifies the development and maintenance of a single code base that can run in three different devices just by setting the desired compilation flag. For the CPU, iGPU and dGPU versions we use the `icpx -fsycl` compiler. Also note in Figure 3.6 that the dGPU `queue` is constructed (in line 7) using a lambda that returns true for the device using the CUDA backend.

```
1 #ifdef SCPU
2     sycl::queue queue{sycl::cpu_selector_v};
3 #elif SiGPU
4     sycl::queue queue{sycl::gpu_selector_v};
5 #elif SdGPU
6     sycl::queue queue{[](auto& d)
7     {return (d.get_platform().get_backend() == sycl::backend::ext_oneapi_cuda)
          ;}};
8 #endif
```

Figure 3.6: Construction of the SYCL queue depending on the desired device (CPU, iGPU or dGPU).

We also target the discrete NVIDIA GPU in two more ways. The first one is with a pure CUDA code that we name CUDA and that is compiled using the CUDA compiler `nvcc`. The second one is a SYCL-CUDA hybrid in which CUDA kernels are called from SYCL leveraging SYCL interoperability features. This version is called S-CUDA (more details are provided in Section 3.4.2).

### 3.4.1. Optimization 1 (O1): GPU oriented tree data structure

We have already discussed the tree data structure used on CPU, which requires pointers and dynamic heap memory allocations and is usually built via recursive functions. On GPUs this same approach is either unfeasible (if recursion and/or dynamic allocation are not allowed) or suboptimal (due to high data divergence). On the contrary, GPU oriented trees are stored in statically created arrays and indices are usually used as pointers to connect parents and children of the tree.

Our approach incorporates ideas that were first used in the graphics domain to solve problems as real-time ray tracing, 3D collision detection, voxel-based global illumination, etc. These ideas were first proposed for the GPU implementation

of Bounding Volume Hierarchies, BVH, by [81] in a method called LBVH. This strategy was later improved by [35] and optimized to increase the GPU occupancy (exploited parallelism) by [71, 72]. We have taken ideas from all of these works and adapted them to work with 2D point clouds instead of 3D objects/primitives (usually triangles). Our method also departures in that we store several points per leaf (instead of one object per leaf) following the MaxNumber strategy mentioned before. MinRadius is discarded on the GPU because it is not suitable for our GPU oriented data structure based on Morton codes that we describe next. We rely on Figure 3.7 that revisits our running example to illustrate our GPU data structure and the steps required to build it assuming a single point per leaf node.

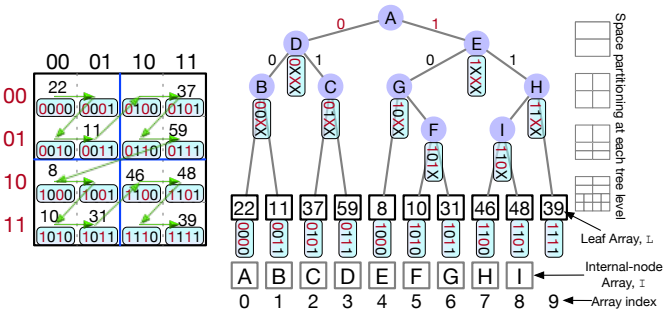



Figure 3.7: GPU oriented data structure: Binary radix tree based on Morton codes.

In order to store a 2D space into a 1D array we use a "space filling curve" that keeps locality thanks to indexing the 2D points with Morton codes [13]. The first step consists in computing the Morton codes associated to the points. This is done by first discretizing the $(x,y)$ coordinates of each point and then interleaving the bits of both coordinates. For example, in the figure, the point with $z = 59$ has a discretized $x = 11$ and $y = 01$ which results in a Morton code $m = 0111$. If we sort the points using as sorting key the Morton codes, we can linearize in a 1D array of leaf nodes, L, the cloud of points maintaining the locality (neighbor points in the 2D space are still close in the 1D array). In our implementation and for the point clouds that we evaluate, we use 32 bits unsigned integers to store the Morton codes which give us 16 bits for each coordinate. For bigger point clouds we could use 64 bit unsigned integers instead. In any case, with this strategy we get an array of leaf nodes in which each node also includes the bounding box of the points stored at each leaf node.

A sorted list of Morton codes has several interesting properties, being the most relevant one that it allows us to build a binary radix tree (BRT) in a single pass.

Figure 3.7 shows the BRT built from the sorted Morton codes of the running example. One of the characteristics of a BRT is that the number of nodes is known in advance and equal to the number of leaves minus one. In our example we have 10 leaves and 9 internal nodes. This is a very important property because it enables us to pre-allocate the node array and also construct the tree in parallel. Each node has to store the indices (pointers) of the children and the parent as well as the bounding box and minimum value of all the points that hang from that node.

When linearizing the tree nodes in the array of internal nodes, `I`, it is key to also exploit locality by keeping related nodes (children, siblings, parent) close in the node array. To this end we leverage some additional properties of the BRT based on Morton codes that we will illustrate with our example in Figure 3.7. We have named the nodes in the node array with the sorted sequence of letters from $A$ to $I$, aligned with the same indices of the sorted list of leaves in the leaf array. Now, to establish the parent-children relationships we use the following rules:

1. Each internal node corresponds to the longest common prefix shared by the keys in its respective subtree, and singleton nodes (nodes with a single child) are avoided. For example, node $D$ is ancestor of points $22, 11, 37$ and $59$, all with Morton codes of the class $0XXX$. There is not an internal node for the class $100X$ because only the point 8 belongs to that class.
2. Each internal node partitions its Morton codes according to the first differing bit (starting from the most significant one). For example, the root node $A$ partitions the Morton codes in two classes: $0XXX$ and $1XXX$. Note that it also partitions the 2D space by the $y$ coordinate.
3. Each internal node has an index in the node array, `I`, that corresponds either to the end of a class (if it is a left child) or to the beginning of a class (if it is a right child). For example, the node $D$ is the left child of $A$ and has the index 3 which corresponds to the point 59 that is the end of the class $0XXX$ in the leaf array, `L`. Similarly, node $E$ is the right child of $A$ and points to the beginning of the class $1XXX$. The root node is always the first node and points to the beginning of the whole array `L`.

With these rules, in parallel, one thread per tree node can find its two children and update the indices/pointers with the parent-child relationship as detailed in [71]. The last step is to compute the bounding boxes and minimum values of the nodes required by the memoization strategy presented in Section 3.3.3. This is done in a single parallel bottom-up pass from the leaves to the root. The bounding box of a node is the union of the bounding boxes of its children and

similarly, the minimum value of a node is computed from the minimum value of its children. The parallel computation starts with a GPU thread per leaf-node that computes the bounding box and minimum for that node. Then the first thread that visits a parent node sets to one an atomic variable initialized to zero and dies. The thread that comes from the other children finds the atomic variable already modified and takes care of updating in the parent node the bounding box and minimum value from the two children.

An additional optimization strategy consists in reducing the depth of the tree by constructing an octree out of the binary tree. The idea is to collapse each block of three tree levels in the binary tree into a single octree node. Note that this new octree partitions the 2D space in contrast with the 3D octree that was described in Section 3.2.2. For example, in Figure 3.7 the nodes $A, B, C, D, E, G$, and $H$, can be collapsed into a single octree root node with 8 children $(22, 11, 37, 59, 8, F, I,$ and $39)$.

### 3.4.2. Optimization 2 (O2): GPU parallelization

As we said, we have three code implementations able to run on the GPU: i) the SYCL implementation that can generate two versions to run either on the iGPU or the dGPU (S-iGPU, S-dGPU); let's not forget that this implementation can build a version to run on the CPU, S-CPU; ii) the CUDA implementation running on the dGPU (CUDA); and the SYCL-CUDA hybrid also running on the dGPU (S-CUDA). In all the cases the kernel that executes the OWM traversal phase is based on the same data-parallel paradigm: the body of the OWM parallel loop (line 2 in Algorithm 2) is implemented as a kernel that is executed by the GPU threads. The SYCL syntax for that implementation is sketched in Figure 3.8(a). In line 1 we invoke the `submit` member function of the `queue` object to submit a command group to the corresponding device. This command group includes a `parallel_for` invocation (line 3) specifying the number of iterations (`NumSteps`) and the lambda function with the kernel code (line 5) that is executed for each iteration, `it`.

The main differences with the SYCL-CUDA hybrid implementation are highlighted in Figure 3.8(b). We first note that the command group calls a `host_task` member function instead of the `parallel_for` one. This enables the execution of the interoperability code required to properly get access to the CUDA kernel arguments and data. After that, we can just call the CUDA kernel as usual (line 5) using the `<<<grid, block>>>` syntax to specify the number of blocks and threads per block, respectively. In the pure CUDA implementation we do

```
queue.submit([&](sycl::handler &h) {
  ... // host-to-device data management
  h.parallel_for(NumSteps,[=](auto it)
      {
    ...
    searchMinSYCL(it, ...)});
});
```

(a)

```
queue.submit([&](sycl::handler& h) {
  ... // host-to-device data management
  h.host_task([=](sycl::interop_handler
       ih) {
    ... // reinterpret with
        interop_handler
    searchMinCUDA<<<grid, block>>>(...)
        ;});
});
```

(b)

Figure 3.8: Submitting a SYCL kernel (a) or a CUDA kernel (b) from SYCL code.

not need the `queue.submit()` nor the `host_task` member functions, as we can just call the CUDA kernel directly from the host code.

### 3.4.3. Optimization 3 (O3): reduce the number of accesses by memoization

Out of the two memoization strategies implemented on CPU and described in Section 3.3.3, only the memoization saving min approach (see Figure 3.4(b)) is applicable to the GPU. The first strategy (memoization exploiting overlap) is based on temporal locality (one thread processing one step of the OWM algorithm can take advantage of the minimum found on the previous step), but on the GPU implementation there is a thread per step, which renders this approach unfeasible. The second strategy (b) is effectively implemented for the GPU OWM traversal, which requires storing at each tree node the number of points and minimum of all the points hanging from that node.

### 3.4.4. Optimization 4 (O4): Tune the Granularity

The GPU architecture is quite sensible to data and control divergence, which translates in that achieving load balance is more relevant on this kind of device than on the CPU. For that reason, keeping a balanced number of points per leaf-node is crucial to achieve good performance. For that reason we have implemented on the GPU an optimized version of the MaxNumber strategy described in Section 3.3.4 for the CPU. The idea is to force all the leaf-nodes of the tree to store exactly MaxNumber points, being MaxNumber a power of 2. This can

be achieved once all the points of the LiDAR cloud have been sorted according to their Morton codes (see Section 3.4.1). Then, the sorted list of points is divided into chunks of MaxNumber points each, and each chunk is assigned to a leaf-node. The data members of each leaf-node are later updated with the information related to the bounding box and minimum of the points it contains. Note that with this static partition of LiDAR points among leaf-nodes, the bounding boxes of sibling nodes can overlap and this has a negative impact on the OWM traversal (that has less opportunities to prune the nodes). However, the alternative implies variable size leaf-nodes, and we found that this irregularity degraded times on the targeted GPU devices, so we did no explore that option further.

## 3.5. Experimental Results

Before analyzing the performance of the different CPU and GPU versions, we first validate the final results of our proposed solution to assess that the implemented optimizations do not affect the quality of the detected ground surface.

### 3.5.1. Validation

We consider as the "gold" ground surface, $G\varphi_{t=0}$, the one that results from running the baseline sequential version of the OWM code. This is the one that is used to validate the results of the other versions studied in this chapter. However, it should be noted that the implementation of some optimizations turns the resulting output into a non-deterministic one. For example, the selection of the minimum value depends on the order in which it is found. Several LiDAR points, with different ids can have the same z-value[6], so different search orders can result in different ids of the minimum. This hampers a direct comparison between the gold version and the results of the different optimized codes.

Another consideration arises from the memoization strategy implemented to optimize the processing of overlapping sliding windows. This approach avoids revisiting points in the overlapped areas between consecutive windows, instead estimating the point count based on the density of the sampled region. While this technique significantly reduces computational overhead, it introduces a degree of approximation that may result in slight variations in the selection LLPs compared to an exhaustive search. This trade-off between performance and precision

[6]This is quite frequent in the LiDAR clouds used in our experiments because the z-value is provided with maximum precision of 0.001 and spatially closed points can have the same altitude.

is characteristic of the optimizations employed in large-scale sensor data processing, where the sheer volume of data necessitates such compromises to achieve practical execution times. The impact of this approximation on the overall accuracy of ground surface detection is carefully evaluated in the following sections, demonstrating that the benefits in processing speed outweigh the minimal deviations in results.

There are several strategies that have been used to estimate the error produced when processing LiDAR point clouds [108, 116]. In [116] absolute error arising from computing digital terrain models are provided and errors in the order of centimeters are accepted. In our work we consider a valid ground point of the resulting ground surface, $\varphi_{t=0}$, if in the nearby (using $o_x/2 = 1$ m as radius) there is a corresponding point in the gold ground surface, $G\varphi_{t=0}$, with a difference in altitude of less than 1 cm. With this criterion, in all our experiments we always obtain more than 97% of valid ground points.

### 3.5.2. Performance analysis on CPU

In this section we discuss the performance improvements that our optimization strategies achieve on our target CPU. We start evaluating optimization 1 (O1) with respect to the baseline algorithm (see sequential tree construction and parallel OWM traversal times -based on OpenMP- in Table 3.1). By default, this baseline uses as base criterion $MinRadius = 0.1$ in the tree construction phase, and we keep this criterion till optimization 4 (O4), where we explore other strategies and values to control the leaf-nodes granularity. After O1, we evaluate the performance improvement that each new optimization adds to the previous one (+O2, +O3, +O4). Our study breaks down the impact of each optimization in the tree construction (Tree C.) and OWM traversal (OWM) phases, as well as in the total execution (Total) times. Table 3.2 summarizes the corresponding improvements for each optimization w.r.t. the previous one, always using 8 threads. In the next subsections, we analyze each optimization in more detail.

#### 3.5.2.1. Optimization 1 (O1): 2D-space bipartition data structure

As we see in Table 3.2, O1 improves the tree construction times by around 50% for all clouds, while parallel OWM traversals are between 1.88x and 2.42x faster. Thus, we see that the change from a 3D bipartition to a 2D bipartition data structure has a greater impact in the traversal phase. More importantly, this optimization takes up to 90% less memory footprint. For instance, using a

| | O1 vs OpenMP baseline | | | O2 vs O1 | | |
|---|---|---|---|---|---|---|
| Cloud | Tree C. | OWM | Total | Tree C. | OWM | Total |
| Alcoy | 1.49x | 1.88x | 1.64x | 6.27x | 1.07x | 2.18x |
| Arzua | 1.52x | 2.42x | 1.80x | 5.52x | 1.23x | 2.65x |
| BrionF | 1.48x | 2.03x | 1.72x | 5.15x | 1.63x | 2.66x |
| BrionU | 1.50x | 1.88x | 1.68x | 4.89x | 1.76x | 2.68x |
| | O3 vs O2 | | | O4 vs O3 | | |
| Cloud | Tree C. | OWM | Total | Tree C. | OWM | Total |
| Alcoy | 0.73x | 3.31x | 1.88x | 4.14x | 2.04x | 2.83x |
| Arzua | 0.83x | 3.68x | 1.73x | 2.49x | 1.39x | 2.00x |
| BrionF | 1.02x | 6.81x | 2.55x | 1.00x | 1.00x | 1.00x |
| BrionU | 1.06x | 6.55x | 2.60x | 1.00x | 1.00x | 1.00x |

Table 3.2: Improvement of each optimization w.r.t. the previous one.

heap profiler (Massif from Valgrind[7]), we found that for the smallest point cloud in our dataset (Alcoy) the octree requieres 1.23 GB whereas the quadtree only 192 MB. For larger datasets we save even more memory space. In any case, the O1 optimization improves total execution times up to 1.8x.

#### 3.5.2.2. Optimization 2 (O2): Parallelization

Here we evaluate the different parallelization strategies (O2) that we propose in Section 3.3.2. All parallel executions run with 8 threads. Let us recall that from now on, we use TBB as programming model to implement our optimizations.

Figure 3.9 shows the times of the parallel tree construction described in subsection 3.3.2.2 for different values of the *lev* parameter (tree level at which the initial part of the tree is created sequentially). As discussed in the mentioned subsection, small values of *lev* enable little parallelism in the final tree construction stage, while large values penalize the first sequential stage because it takes longer, but also the deeper the tree the more leaf-nodes are created, so a trade-off value must be selected for the corresponding cloud and number of threads. In the figure we see that for our clouds and 8 threads, the optimal *lev* is between 5 and 7. Table 3.3 represents some interesting features related to the trees created at different levels. Recall from Table 3.1 that Alcoy and Arzua cover a larger area but with less points than BrionF and BrionU clouds, and therefore the latter have a higher density of points. In clouds with a lower density (Alcoy, Arzua),

[7] https://valgrind.org/docs/manual/ms-manual.html

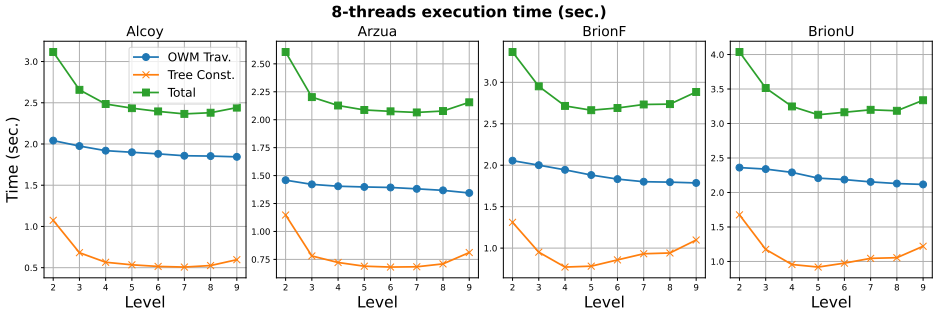



Figure 3.9: Optimization O2: tree construction, OWM traversal and total times depending on $lev$ using 8 threads.

increasing $lev$ practically does not affect the number of leaf-nodes, because there is already a high ratio of empty leaf-nodes at $lev = 5$, and therefore the cost for creating the corresponding temporary vector of points in each non-empty node is similar. However, in clouds with a higher density, increasing $lev$ also increases the number of leaf-nodes and therefore the cost for creating the corresponding temporary vector of points for each non-empty leaf-node.

| Cloud | % empty leaf-nodes | ratio $no_7/no_5$ | ratio $no_9/no_5$ |
|---|---|---|---|
| Alcoy | 51% | 1.00 | 1.01 |
| Arzua | 11% | 1.00 | 1.01 |
| BrionF | 2% | 1.08 | 1.19 |
| BrionU | 2% | 1.08 | 1.20 |

Table 3.3: Tree features for each cloud: Percentage of empty leaf-nodes at $lev = 5$, number of leaf-nodes at $lev = 7$ vs. $lev = 5$, and number of leaf-nodes at $lev = 9$ vs. $lev = 5$.

Figure 3.9 also shows the times of our TBB OWM traversal version described in subsection 3.3.2.1 for different values of $lev$. Now, we notice that increasing the depth of the tree slightly decreases the parallel traversal times, because increasing the depth decreases the size of the leaf-nodes and thus the number of check operations per node. From the figure we see that the values of $lev$ that achieve the optimal total execution times are 7, 7, 5, and 5 for Alcoy, Arzua, BrionF and BrionU, respectively. For these values, the parallel TBB OWM traversal incrementally improves OpenMP OWM traversal from 1.07x to 1.76x. More interestingly, for the same $lev$ values the parallel tree construction strategy has a more significant impact on performance: up to a 6.27x of improvement. In any

case, the O2 optimization incrementally improves the total execution times up to 2.68x w.r.t. O1 (see Table 3.2).

#### 3.5.2.3. Optimization 3 (O3): reduce the number of accesses by memoization

In this subsection we evaluate the impact of the memoization strategies (O3) proposed in Section 3.3.3 in our parallel codes (parallel executions with 8 threads again). Figure 3.10 shows the times of the parallel tree construction, parallel OWM traversal and total time for different values of the *lev* parameter, which we explore again when we apply all the memoization strategies proposed in the mentioned section. As expected, there is a trade-off *lev* value that optimizes the times: 7, 7, 4 and 4 for Alcoy, Arzua, BrionF and BrionU, respectively. These *lev* values are used to report the O3 improvements w.r.t. O2 that we see in Table 3.2. Now, the tree construction phase takes longer because building the augmented tree with the memoization feature requires to add two new data members to each tree node: the minimum and the number of points hanging from the node. As we notice in Table 3.2, the tree construction times can degrade up to 27%. However, O3 has a more profound impact on the OWM traversal times, which improve from 3.31x to 6.81x. We should note that we achieve higher incremental improvements in the clouds with higher density. We have also factored out the impact that each memoization strategy achieves in the traversal times: strategy (a) - memoization exploiting overlap - improves the parallel traversal times up to 1.76x, while strategy (b) - memoization saving min - is the one with the highest impact, because it further improves these times up to 2.93x.

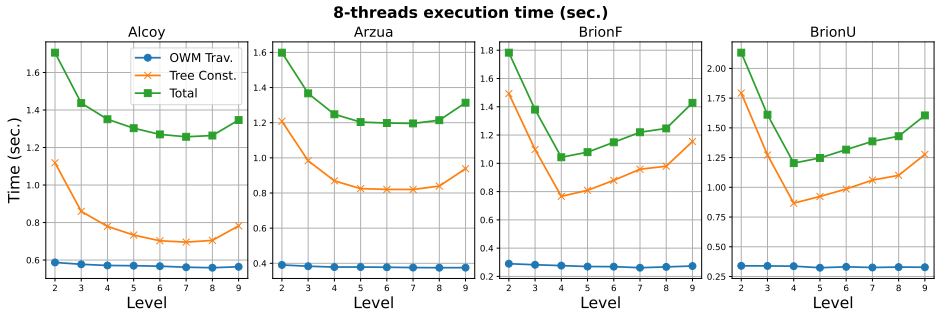



Figure 3.10: Optimization O3: tree construction, OWM traversal and total times depending on *lev* using 8 threads.

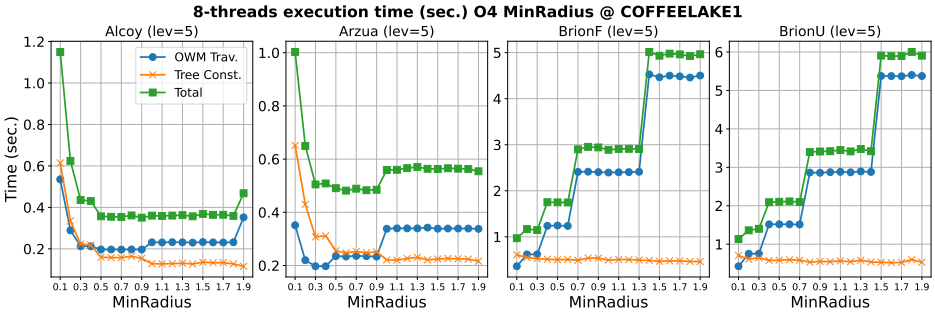
Interestingly, Figure 3.10 tells us that the tree construction phase represents

now the bottleneck in the total execution time. In any case, thanks to the combination of the memoization strategies (a) and (b) and the resulting reduction in the memory bandwidth requirements, O3 optimization incrementally improves the total execution times up to 2.6x (see Table 3.2).

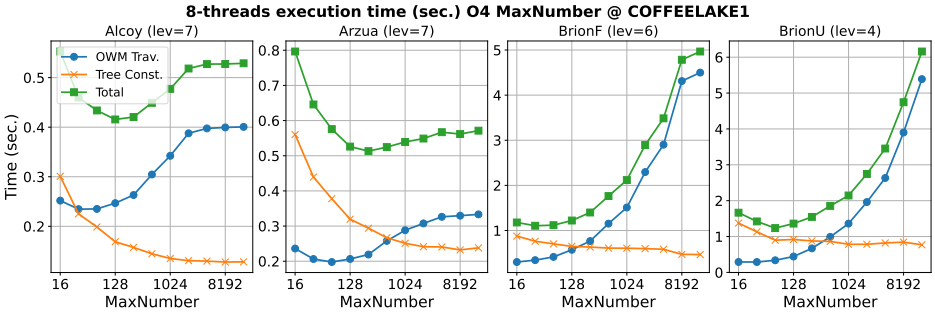
#### 3.5.2.4. Optimization 4 (O4): Tune the Granularity

In this section we evaluate the impact of the strategies for tuning the granularity of the tree leaf-nodes, which we propose in Section 3.3.4: MinRadius and MaxNumber. For both strategies we have explored again the *lev* parameter, and in Figure 3.11 we show the parallel times (tree construction phase, OWM traversal phase and total times) for the optimal *lev* value found for each cloud and strategy (indicated in the figure). In particular, Figure 3.11(a) represents the times for MinRadius when this parameter varies between 0.1 and 0.9, whereas Figure 3.11(b) shows the times for MaxNumber when this parameter goes from 32 to 65536. Four main conclusions can be extracted from the figures:

- The tree construction times decrease when MinRadius or MaxNumber increases. Clearly, increasing the granularity of the leaf-nodes decreases the depth of the tree and the number of intermediate nodes created, what have an impact in the construction times. For this phase, the optimal MinRadius times are up to 33% more efficient than MaxNumber times.
- However, OWM traversal times tend to increase when MinRadius (higher than 0.2) or MaxNumber increases. Now, increasing the granularity of the leaf-nodes requires to invest more time checking all the points stored in these nodes. Again, for this phase, optimal MinRadius times are more efficient than optimal MaxNumber times: up to 17% faster. Interestingly, more sparse clouds (Alcoy, Arzua) show little degradation of times when increasing the granularity, contrary to more dense clouds (BrionF, BrionU).
- Therefore, for each strategy and cloud there is a sweet point at which the total time is optimal. The MinRadius and MaxNumber values that achieve that optimal for each cloud are shown in Table 3.4, along with the *lev* parameter. In any case, the optimal MinRadius total times always outperform the optimal MaxNumber ones: from 1.08x to 1.17x. We have found that MinRadius generates shallower and wider trees populated with smaller leaf-nodes than MaxNumber, and that type of tree is better exploited (creation and traversal) on our target CPU.
- With these results augmented by some additional experimentation we can recommend as a rule of thumb to use level=5 and minRad=0.3, resulting in a maximum performance loss of 14.63% compared to the optimal




(a) MinRadius changes between 0.1 and 1.9




(b) MaxNumber changes between 16 and 16584 in powers of 2

Figure 3.11: Optimization O4: tree construction, OWM traversal and total times using 8 threads.

configuration for each individual cloud. In addition, we should not fright about getting an approximation around these recommended values since the potential performance loss is not significant in most cases.

| | MinRadius | | MaxNumber | |
|---|---|---|---|---|
| Cloud | *lev* | *MinRad* value | *lev* | *MaxNum* value |
| Alcoy | 5 | 0.9 | 6 | 128 |
| Arzua | 5 | 0.6 | 6 | 256 |
| BrionF | 4 | 0.1 | 4 | 64 |
| BrionU | 4 | 0.1 | 4 | 64 |

Table 3.4: Optimization O4: Optimal parameters for strategies MinRadius and MaxNumber.

Once we have selected the optimal strategy, MinRadius, and tuned the optimal

granularity for each cloud, we see in Table 3.2 the impact that the O4 optimization has in the tree construction, OWM traversal and total times when compared to O3 with MinRadius=0.1. The incremental improvement in performance is higher in clouds with lower density (Alcoy, Arzua): up to 2.83x. However, in clouds with higher density (BrionF, BrionU) the incremental improvement is nonexistent because the baseline MinRadius=0.1 ends up being the best one.

| Strategy | MinRadius | | | | MaxNumber | | | |
|---|---|---|---|---|---|---|---|---|
| Cloud | Alcoy | Arzua | BrionF | BrionU | Alcoy | Arzua | BrionF | BrionU |
| Optimal Value | 0.9 | 0.6 | 0.1 | 0.1 | 128 | 256 | 64 | 64 |
| $\hat{X}$ | 19 | 79 | 8 | 8 | 52 | 75 | 19 | 19 |
| Observed Density | 13.1 | 40.7 | 152.9 | 163.7 | 24.6 | 94.6 | 207.8 | 234.3 |
| Area | 8.09 | 3.81 | 0.21 | 0.23 | 11.19 | 2.76 | 0.33 | 0.32 |
| # empty leaf-nodes | 5358 | 31 | 50766 | 45816 | 4274 | 4 | 4879 | 4895 |
| Tree Level | # leaf-nodes | | | | | | | |
| 6 | 2266 | 0 | 1593 | 1664 | 2274 | 0 | 1606 | 1680 |
| 7 | 186 | 0 | 318 | 295 | 186 | 0 | 347 | 321 |
| 8 | 564 | 0 | 812 | 823 | 550 | 6144 | 1024 | 966 |
| 9 | 758 | 262144 | 2451 | 2380 | 57962 | 206435 | 5489 | 4760 |
| 10 | 444520 | 0 | 8612 | 7885 | 198152 | 116737 | 287377 | 240292 |
| 11 | 0 | 0 | 2.4e+06 | 2.3e+06 | 62244 | 30777 | 1.1e+06 | 1.3e+06 |
| 12 | 0 | 0 | 0 | 0 | 2635 | 1596 | 53807 | 146101 |
| 13 | 0 | 0 | 0 | 0 | 142 | 63 | 260 | 620 |
| 14 | 0 | 0 | 0 | 0 | 24 | 4 | 0 | 0 |
| Total | 448294 | 262144 | 2.4e+06 | 2.3e+06 | 324169 | 361756 | 1.5e+06 | 1.7e+06 |

Table 3.5: Depth tree, mode ($\hat{X}$) of number of points per leaf-node, Observed density [$pts/m^2$], average Area [$m^2$] of leaf-node, number of empty leaf-nodes, number of leaf-nodes at each tree level, and total number of leaf-nodes.

Figure 3.12 shows the distribution of the sizes of the leaf nodes for a range of different values of MinRadius and MaxNumber, selecting the configuration with the best total execution time for each cloud as 'Opt.' (Optimal Value in Table 3.5) and bringing the plot to the foreground. From the figure, we observe that the optimal MinRadius generates taller and narrower shapes for denser clouds, and wider and shallower shapes for sparser clouds. This behavior is natural if we aim to balance the load in the OWM traversal phase. The mode ($\hat{X}$) is a small value, ranging from 8 to 79 points per leaf node in the MinRadius case, and

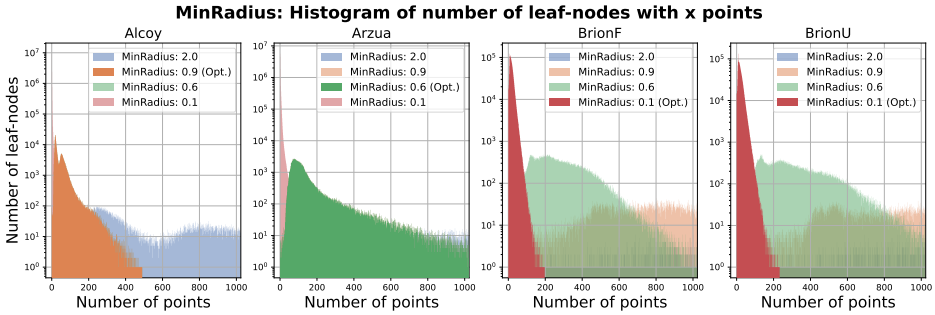
from 19 to 75 in the MaxNumber case, as shown in Table 3.5. This corroborates the idea that the optimal granularity is the one that better balances the load in the OWM traversal phase. In other words, we desire small leaf nodes to better exploit parallelism, but not so small that we incur the overhead of traversing many nodes. It is also interesting to note that the histograms for MinRadius are more widely distributed than those for MaxNumber, as the latter imposes a fixed maximum number of points per leaf node. This observation might lead one to think that MinRadius is a worse strategy, potentially unbalancing the load in the OWM traversal phase. However, MinRadius is the strategy that achieves the best total execution times. Regarding the number of leaf nodes with a point count around the mode ($\approx y(\hat{X})$), we see that the optimal MinRadius generates more leaf nodes with a point count around the mode compared to the optimal MaxNumber. This is a good indicator that the OWM traversal phase will be more balanced when using MinRadius. Table 3.5 further corroborates this finding: MinRadius generates wider trees populated with more, smaller leaf nodes than MaxNumber. This is supported by the smaller densities and areas measured in the leaf nodes when using MinRadius.

#### 3.5.2.5. Summary of each optimization and scalability anlysis on CPU

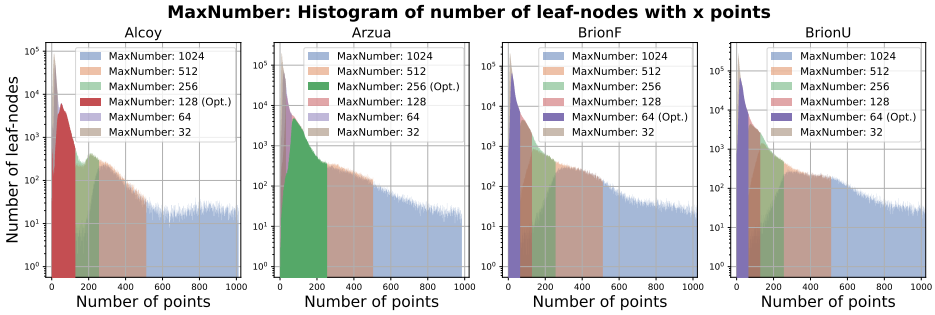
Table 3.6 summarizes the improvement factor of each optimization w.r.t. the parallel OpenMP baseline implementation (see Table 3.1), for tree construction, OWM traversal and total execution times. Note that optimizations are cumulative, this is, optimization 2 (O2) also includes optimization 1 (O1), and optimization 3 (O3) encompasses the two previous ones. We can see larger improvements thanks to O3 in the OWM traversal phase, which is originally the most time-consuming part of the algorithm, but with this optimization the tree construction can become the bottleneck. On the other hand, O2 and O4 also show significant improvements for the tree construction times.

Figure 3.13 shows the relative weight of each optimization over the OpenMP baseline (in %) for the tree construction, OWM traversal and total times. This figure gives us some interesting insights about the impact of the optimizations regarding the density of the cloud. We see that the clouds with a lower density of points, Alcoy and Arzua, are more affected by optimization O4 (thanks to the relevant impact of this optimization in their tree construction phase), while the clouds with a higher density, BrionF and BrionU, are more affected by optimization O3 (thanks to its significant impact on the OWM traversal phase).

Finally, Figure 3.14 shows the parallel speedup of the O4 TBB implementation for different number of cores. We also break down the speedup numbers for the




(a) Histogram of the number of points per leaf node for MinRadius values of 2.0, 0.9, 0.6, and 0.1, plotted for each of the tested clouds. The y-axis uses a logarithmic scale to better compare the different distributions corresponding to each MinRadius value.




(b) Histogram of the number of points per leaf node for MaxNumber values of 1024, 512, 256, 128, 64, and 32, plotted for each of the tested clouds. The y-axis uses a logarithmic scale to better compare the different distributions corresponding to each MaxNumber value.

Figure 3.12: Histogram with the frequency of leaf-nodes for each number of points.

tree construction and the OWM traversal phases. While the tree construction scales for all the clouds (although parallel efficiency drops below 60% on 8 cores), the traversal phase does not scale after 6 cores for the dense point clouds. Note that after all the implemented optimizations the scalability of this phase has hardly improved (see the 8 cores speedup of the OpenMP baseline implementation in the last column of Table 3.1). This is, both the sequential and the 8 cores execution times have improved at the same ratio (more than 11×). Although our optimizations have reduced the memory footprint and bandwidth requirements, the problem is still highly memory bound and in some cases, mainly for the denser Brion's clouds, where the trees are larger and deeper, adding cores does not help. In subsection 3.5.3.5 we collect different metrics from the roofline model

| Cloud | O1 Tree | O1 OWM | O1 Total | O2 Tree | O2 OWM | O2 Total |
|---|---|---|---|---|---|---|
| Alcoy | 1.49x | 1.88x | 1.64x | 9.33x | 2.01x | 3.58x |
| Arzua | 1.52x | 2.42x | 1.80x | 8.41x | 2.97x | 4.77x |
| BrionF | 1.48x | 2.03x | 1.72x | 7.64x | 3.31x | 4.58x |
| BrionU | 1.50x | 1.88x | 1.68x | 7.36x | 3.32x | 4.51x |

| Cloud | O3 Tree | O3 OWM | O3 Total | O4 Tree | O4 OWM | O4 Total |
|---|---|---|---|---|---|---|
| Alcoy | 6.81x | 6.65x | 6.74x | 28.20x | 13.54x | 19.09x |
| Arzua | 7.01x | 10.93x | 8.24x | 17.47x | 15.19x | 16.44x |
| BrionF | 7.79x | 22.55x | 11.70x | 7.79x | 22.55x | 11.70x |
| BrionU | 7.80x | 21.75x | 11.71x | 7.80x | 21.75x | 11.71x |

Table 3.6: Improvement over the OpenMP baseline (see Table 3.1) for each optimization.

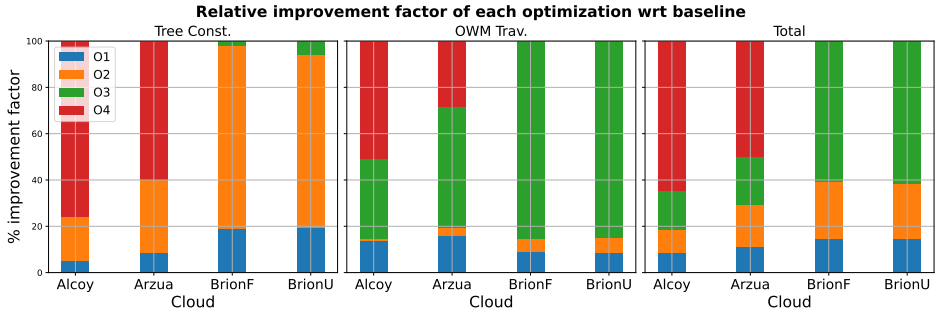



Figure 3.13: Relative improvement factor of each optimization over the OpenMP baseline (%) using 8 threads.

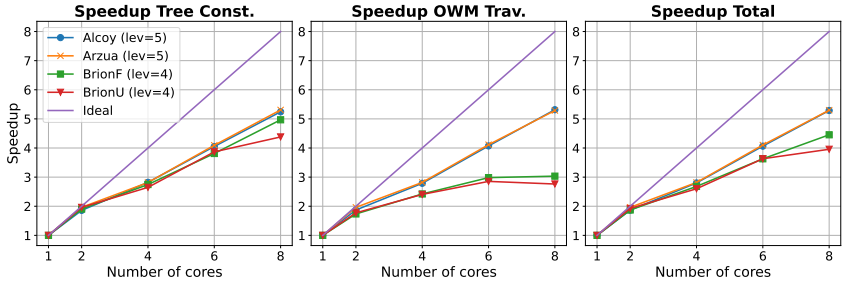



Figure 3.14: Speedup plot for different core/thread counts.

to discuss this issue in further detail.

### 3.5.3. Performance analysis on GPU

In this section we study now the impact of the GPU optimizations proposed in Section 3.4. For the evaluation we use now as reference TBB base, i.e., the optimal CPU times obtained after applying all optimizations discussed in the previous section (O1 to O4) to the TBB implementation using 8 threads, and compare them with the different GPU implementations described in Section 3.4.2: SYCL on iGPU and dGPU devices (S-iGPU, S-dGPU), and CUDA on the dGPU device (NVIDIA discrete GPU). We also include the SYCL implementation running on the CPU device (S-CPU). Table 3.7 summarizes the portability across devices for our implementations.

| Prog. model / device | CPU | i-GPU | d-GPU |
|---|---|---|---|
| TBB | ✓ (TBB base ) | ✗ | ✗ |
| SYCL | ✓ (S-CPU) | ✓ (S-iGPU) | ✓ (S-dGPU) |
| CUDA | ✗ | ✗ | ✓ (CUDA) |

Table 3.7: Portability across devices for our implementations.

#### 3.5.3.1. Optimizations 1 and 2 (O1 & O2): 2D-space bipartition and parallelization

Here we consider the impact of the 2D-space bipartition and parallelization optimizations in the GPU execution times, because these two features come together naturally and are actually a must in an efficient GPU implementation, as explained in Sections 3.4.1 and 3.4.2. The breakdown of the tree construction and OWM traversal times are shown in Figure 3.15 for the CPU versions: TBB base and the SYCL CPU, S-CPU, as well as for the GPU versions: S-iGPU, S-dGPU, and CUDA. The SYCL and CUDA results do not include optimizations O3 (memoization) and O4 (tuning granularity) that will be studied in next subsections. In these results we consider as base criterion MaxNumber=512 points per leaf-node.

Four main conclusions can be extracted from Figure 3.15:

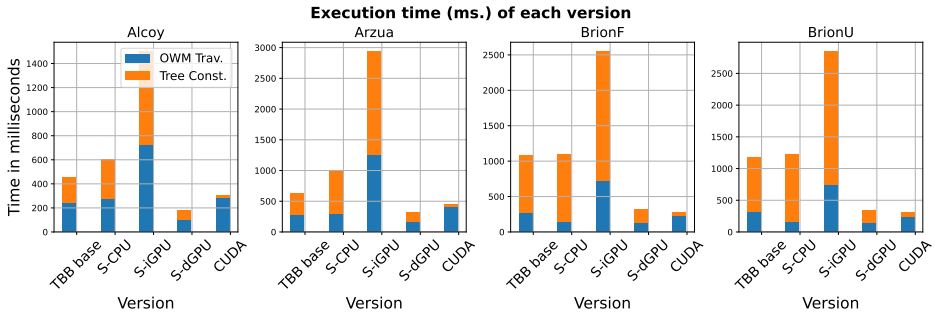



Figure 3.15: Optimizations O1 & O2: Execution times comparison of TBB base, SYCL and CUDA implementations, without memoization and MaxNumber=512.

- The CUDA implementation running on the dGPU is the fastest for Brion's clouds, but the second fastest for Alcoy and Arzua. The total execution times improve from 1.39x to 3.56x when compared to the TBB base. Notably, tree construction times show the highest improvement, ranging from 11.34x (Alcoy) to 14.23x (BrionF). However, it is worth noting that the OWM traversal times for CUDA show a slight degradation of around 4% compared to the TBB base traversal times. It can be inferred from this result that the new tree construction phase, where the GPU oriented data structure is built, effectively takes advantage of the device architecture.
- From the GPU SYCL implementation (S-iGPU, S-dGPU), S-iGPU performs the worst due to the lower computational capabilities of the integrated GPU. On the other hand, S-dGPU is the fastest implementation for Alcoy and Arzua clouds, even faster than the native CUDA implementation. It achieves improvements of 2.49x and 1.80x, respectively. In the case of the Brion's clouds, S-dGPU achieves improvements of 2.90x and 2.86x for BrionF and BrionU, respectively, but it does not improve the CUDA implementation total times. Interestingly, in contrast to the CUDA implementation, S-dGPU shows a great improvement in the OWM traversal times that range from 1.41x to 2.27x when compared to TBB base.
- The efficiency of the SYCL implementation can be evaluated by comparing S-dGPU and CUDA. The SYCL abstraction layer introduces only a 4% degradation in the total execution time compared to CUDA on average, with slightly improved execution times for the Alcoy and Arzua clouds. The main source of inefficiency in the SYCL version lies in the tree construction phase, which can be up to 4 times slower than CUDA. This is attributed to the inefficient radix-sort operation called twice in our code, which cannot

compete with the highly optimized CUDA counterpart specifically tailored to the NVIDIA architecture. However, as mentioned before, the OWM traversal times show a notable improvement over CUDA for all clouds, ranging from 25% to 61% faster. Despite the fixed scenario of MaxNumber=512 without memoization, which can lead to higher warp divergence on the NVIDIA architecture, the SYCL driver handles this inconvenience more effectively.

- The portability of the SYCL implementation can be analyzed by comparing TBB base and S-CPU. Let's recall that the data structure constructed in TBB is based on a quadtree, whereas the SYCL version builds a radix tree based on Morton codes, so these phases are not comparable, because while the second implementation exhibits higher parallelism, it does so by increasing the computational cost. However, the OWM traversals are similar, although memoization and tuning of granularity optimizations have not yet been applied in the SYCL implementation. Even though, for this phase, the SYCL implementation is already on average 25% faster than TBB base, thanks precisely to the less memory pressure that the traversal of the SYCL data structure offers and because the SYCL compiler does a better job exploiting the CPU SIMD units.

Due to the poor performance of the S-iGPU version, we skip this one in the rest of our analysis in the next subsections, unless otherwise noted.

#### 3.5.3.2. Optimization 3 (O3): reduce the number of accesses by memoization

In this subsection we study the impact of the memoization strategy presented in Section 3.4.3, where we highlight again that only the saving min approach is applicable (case (b)). OWM traversal times are reduced thanks to this optimization. Table 3.8 shows the OWM traversal times before and after the memoization strategy has been applied. As we see, the incremental improvement (O3 vs O2) is higher in the clouds with higher density for all versions. Specifically, for the SYCL versions, we observe up to a 17% and 53% gain for S-CPU and S-dGPU, respectively. The CUDA version also benefits from memoization, with improvements up to 13%. Overall, this optimization accounts for up to 12% and 5% improvement in the total execution times for S-dGPU and CUDA, respectively, while providing moderate improvements for the S-CPU version.

| | S-CPU | | | S-dGPU | | | CUDA | | |
|---|---|---|---|---|---|---|---|---|---|
| Cloud | NM | M | O3 vs O2 | NM | M | O3 vs O2 | NM | M | O3 vs O2 |
| Alcoy | 238.8 | 235.5 | 1.01x | 108.2 | 106.2 | 1.02x | 279.7 | 282.1 | 0.99x |
| Arzua | 285.3 | 252.2 | 1.13x | 204.1 | 172.4 | 1.18x | 419.3 | 407.5 | 1.03x |
| BrionF | 158.0 | 119.7 | 1.32x | 184.2 | 125.3 | 1.47x | 245.5 | 219.9 | 1.12x |
| BrionU | 174.7 | 128.6 | 1.36x | 208.6 | 135.9 | 1.53x | 268.2 | 237.5 | 1.13x |

Table 3.8: Optimization O3: OWM traversal times in msec. with (M) and without memoization (NM) and improvement factor (O3 vs O2).

#### 3.5.3.3. Optimization 4 (O4): Tune the Granularity

Here we evaluate the impact of the strategy for tuning the granularity of the tree leaf-nodes (see Section 3.4.4) in the GPU implementations. For it, we explore all possible values for the MaxNumber parameter: from 4 to 1024, and in Table 3.9 we report the MaxNumber value (MaxN) that gives the best performance in each cloud. Using that optimal value, we also report the incremental improvement (Total) for both the tree construction and the OWM traversal phases for S-CPU, S-dCPU and CUDA.

| | S-CPU | | | | S-dGPU | | | | CUDA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cloud | MaxN | Tree | OWM | Total | MaxN | Tree | OWM | Total | MaxN | Tree | OWM | Total |
| Alcoy | 64 | 0.97x | 1.16x | 1.05x | 64 | 0.89x | 1.28x | 1.08x | 16 | 0.77x | 2.60x | 2.26x |
| Arzua | 128 | 0.98x | 1.35x | 1.06x | 64 | 0.90x | 1.47x | 1.13x | 32 | 0.85x | 2.80x | 2.36x |
| BrionF | 128 | 1.00x | 1.49x | 1.05x | 128 | 0.95x | 1.61x | 1.14x | 32 | 0.88x | 2.47x | 1.81x |
| BrionU | 128 | 1.01x | 1.43x | 1.05x | 128 | 0.97x | 1.59x | 1.15x | 32 | 0.90x | 2.45x | 1.80x |

Table 3.9: Optimization 4 (O4): Improvement factor (O4 vs O3) using the best MaxNumber (MaxN) in O4, in the tree construction (Tree), OWM traversal (OWM) and total times (Total).

Table 3.9 shows that the optimal MaxN parameter is smaller than the default value of 512 used in the previous optimization studies. This leads to deeper data structures and, consequently, a more costly tree construction phase. However, the degradation of the tree construction times is relatively small, although more noticeable for S-dGPU and CUDA, where even smaller MaxN values are found to be optimal. The discrete GPU device is more likely to reduce warp divergences and improve memory coalescence with these smaller MaxN values. Optimization O4 has a more significant impact on the OWM traversal phase, particularly on the versions running on the discrete GPU, where the improvement reaches up to 1.61x for S-dGPU and 2.80x for CUDA. In summary, optimization O4 accounts for up to 6%, 15%, and 136% incremental improvement in the total execution times for S-CPU, S-dGPU, and CUDA, respectively, with the most substantial

gains observed in the CUDA version.

#### 3.5.3.4. Overall improvement factor of SYCL and CUDA implementations

Figure 3.16 summarizes the total improvement of the SYCL and CUDA implementations when all optimizations have been incorporated (from O1 to O4) with respect to the best TBB base code. In the figure, we also include S-CUDA, a hybrid SYCL-CUDA implementation that embeds CUDA kernel calls inside the SYCL code using the SYCL interoperability features (see Section 3.4.2) and run on the discrete GPU.

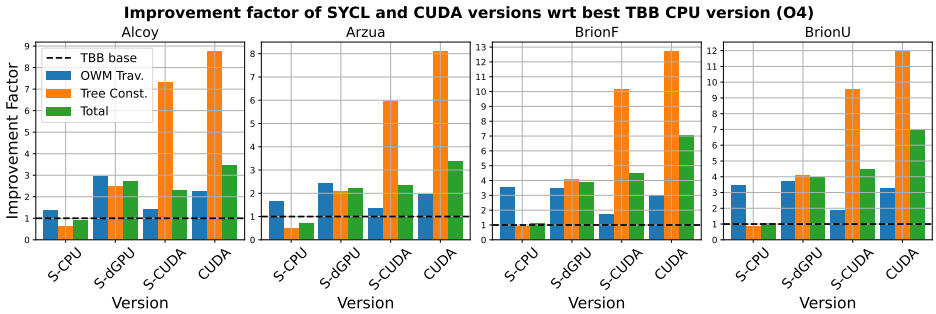



Figure 3.16: Total improvement factor of SYCL and CUDA implementations w.r.t. best TBB CPU base.

As we see in Figure 3.16, after applying all optimizations we outline some relevant findings:

- Optimizations O3 and O4 improve OWM traversal times in both SYCL and CUDA implementations, as explained earlier. However, the S-CPU version does not benefit from the new GPU-oriented tree construction implementation, which aligns with the discussion in subsection 3.5.3.1. The SYCL implementation exhibits higher parallelism but incurs increased computational complexity, resulting in a 28% average performance degradation w.r.t. the CPU oriented TBB implementation. With this, the tree construction is the bottleneck in the CPU (TBB baseline and S-CPU), whereas the OWM traversal is the bottleneck in the CUDA versions (S-CUDA and CUDA), leaving the S-dGPU version between the other two cases, achieving an improvement balance between the tree construction and the OWM traversal phases.

- If you would rather target a GPU device (be it from NVIDIA, Intel, or AMD), SYCL is a viable choice in our case study. For the NVIDIA device, the native CUDA implementation still delivers the fastest performance, with speedups ranging from 3.36x to 7.05x compared to TBB on the CPU. However, the SYCL S-dGPU implementation is quite competitive, outperforming CUDA in the OWM traversal by 21% on average. Despite being 36% slower than CUDA in terms of total execution time, the SYCL implementation offers the advantage of portability across different GPU vendors. Developers who prioritize code portability, and wish to avoid vendor lock-in, may find the SYCL implementation more appealing, even with the performance trade-off.
- If on the contrary a multicore CPU suits your needs, your GPU oriented SYCL implementation is valid as well. The SYCL S-CPU implementation outperforms the best TBB by 150% in the OWM traversal. However, the S-CPU implementation experiences a slight degradation in overall performance, with an average 5% increase in the total execution time. Note that S-CPU is more competitive with denser clouds (Brion).
- S-CUDA suffers from the degradation induced by the runtime of SYCL when compared to the pure CUDA version. The SYCL runtime introduces two sources of overhead: the invocation of the CUDA kernels via the interoperability functionality, and the data movement from/to host-device through SYCL functions. This degradation accounts for around 50% of the total time. Anyway, the S-dGPU can be faster than S-CUDA (see Alcoy) and only 3% slower than S-CUDA on average (attributed primarily to the more efficient CUDA-based tree construction phase). For that reason, we only recommend an S-CUDA implementation as an intermediate step in the translation of an existing CUDA code to SYCL.

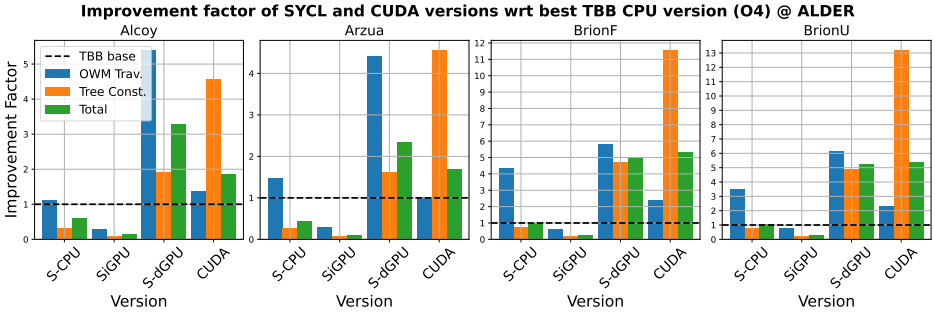



Figure 3.17: Total improvement factor of SYCL and CUDA implementations w.r.t. best TBB CPU base performed on the `Alder` platform.

For further support these conclusions and to deepen the analysis, we have also tested the SYCL and CUDA implementations on the `Alder` platform. The results are shown in Figure 3.17, where we can see the improvement factor over the best TBB CPU base implementation (again, O4 MinRadius) for the `Alder` platform. The conclusions are consistent with those obtained on the `Coffee` plarform, but we can appreciate some important differences:

- The tree construction phase continues to be the bottleneck in the SYCL implementation, and the OWM traversal phase is the bottleneck in the CUDA versions. In the `Alder` platform, the S-dGPU version shows a more balanced improvement between both phases, except for Alcoy and Arzua clouds.
- The S-dGPU implementation is the most competitive in the `Alder` platform, with an average improvement of 3.97x over the TBB CPU base. The CUDA implementation is also competitive, with a performance degradation of 16% on average compared to the S-dGPU version.
- The S-CPU implementation is more competitive than the TBB CPU base in the OWM traversal phase, with an average improvement of 2.61x, carried out specially in the Brion clouds. However, the tree construction phase is less efficient, leading to an average performance degradation of 24% in the total execution time.

These findings suggest that the SYCL programming model, as demonstrated in our testbed, is approaching the "performance portability dream". A single SYCL implementation, when compiled for two distinct devices (S-CPU and S-dGPU), yields competitive execution times compared to highly optimized CPU-oriented (TBB) and GPU-oriented (CUDA) implementations. This progress underscores the potential of SYCL in bridging the gap between different heterogeneous computing architectures.

Returning to the `Coffee` platform, we provide a comprehensive summary in Table 3.10 of the total performance improvements achieved by the SYCL and CUDA implementations relative to the original OpenMP baseline using 8 threads. These results offer a direct comparison with the TBB improvements presented in Table 3.6. The optimized CPU version (TBB with O4) demonstrates impressive processing capabilities, handling between 40 to 68 million LiDAR points per second. However, the GPU implementation (CUDA) substantially surpasses this, achieving a remarkable throughput range of 153–285 million points per second. This represents a significant advancement in the field of massive sensor-based analytics, particularly when compared to previous GPU-based approaches. For instance, the SGF method reported in [55] processed only 3 million LiDAR points

per second on GPU.

| | S-CPU | | | S-dGPU | | |
|---|---|---|---|---|---|---|
| Cloud | Tree | OWM | Total | Tree | OWM | Total |
| Alcoy | 14.02x | 20.56x | 16.30x | 55.02x | 44.95x | 50.08x |
| Arzua | 8.08x | 23.50x | 11.11x | 34.35x | 34.68x | 34.48x |
| BrionF | 6.57x | 84.41x | 12.69x | 29.57x | 82.94x | 44.58x |
| BrionU | 6.63x | 80.99x | 12.82x | 32.00x | 86.33x | 47.83x |
| | S-CUDA | | | CUDA | | |
| Cloud | Tree | OWM | Total | Tree | OWM | Total |
| Alcoy | 160.28x | 21.81x | 42.28x | 191.99x | 34.26x | 63.49x |
| Arzua | 98.03x | 19.21x | 36.24x | 133.70x | 28.03x | 52.10x |
| BrionF | 73.62x | 40.57x | 51.62x | 92.31x | 72.33x | 80.65x |
| BrionU | 74.20x | 43.12x | 53.81x | 92.96x | 76.21x | 83.33x |

Table 3.10: Total improvement factor of SYCL and CUDA implementations w.r.t. OpenMP CPU baseline.

#### 3.5.3.5. Limiting factors and power efficiency considerations for TBB, SYCL and CUDA implementations

In order to better understand the performance bottlenecks for each implementation (TBB, SYCL and CUDA) and how much performance is left on the table because of them on each device, we carried out a roofline analysis for TBB and S-CPU using the Intel Advisor profiler, as well as for CUDA using the NVIDIA Nsight tool. We could not get the S-dGPU data because Nsight does not support SYCL. Table 3.11 shows the roofline data for the function that represents the hotspot in the tree creation (Tree) and traversal (OWM) phases, respectively. From the table, we see that the Arithmetic Intensity (AI) metric confirms the memory bound nature of the algorithm in the different implementations: almost all the analyzed functions are capped by the memory system (either the L3 cache or the main memory -DDR4 on the CPU, GDDR6 on the discrete GPU-). Only the tree creation phase of the CUDA implementation is compute bound (capped by the FP32 unit) for our discrete GPU. We also show the Attainable Performance (AP) for the corresponding AI/Cap because that is the peak performance that each function could achieve in the corresponding device. Then we show the achieved Performance (Perf) and the headroom to the attainable Peak performance (%2Peak) for a sparse and a dense cloud (Alcoy, BrionU), respectively.

One important insight that AI gives is that the implementation targeting the GPU (CUDA) reduces significantly the traffic with memory compared to

TBB, and therefore it increases the room for better performance. However, the traversal in this implementations is capped by the main memory ceiling, what hints that there could be explored more aggressive strategies for exploiting the memory hierarchy in the traversal of our binary radix tree based on Morton codes. On the other hand, the traversal of the quadtree in the TBB implementation already partially exploits cache hierarchy (it is capped by L3). Still, the measured performance of the TBB traversals in our clouds is far from the peak and the %2Peak metric says that there is room for locality improvement, in particular for the dense data clouds.

| | TBB | | S-CPU | | CUDA | |
|---|---|---|---|---|---|---|
| Metric | Tree | OWM | Tree | OWM | Tree | OWM |
| AI (FLOP/Byte) | 0.003 | 0.071 | 0.075 | 0.41 | 0.85 | 0.10 |
| Cap (mem./compute) | L3 | L3 | L3 | DDR4 | FP32 | GDDR6 |
| AP (GFLOPS) | 1.88 | 40 | 42 | 15.7 | 165 | 32 |
| | Alcoy | | | | | |
| Perf (GFLOPS) | 0.23 | 4.67 | 5.2 | 8.68 | 104 | 4.6 |
| %2Peak | 87% | 88% | 87% | 44% | 40% | 85% |
| | BrionU | | | | | |
| Perf (GFLOPS) | 0.2 | 1.27 | 5.76 | 14.3 | 104 | 6.4 |
| %2Peak | 89% | 96% | 86% | 9% | 40% | 80% |

Table 3.11: Roofline metrics for TBB, S-CPU and CUDA: Arithmetic Intensity (AI), memory/compute ceiling (Cap) and Attainable Performance (AP); for Alcoy and BrionU: Performance (Perf) and headroom to the attainable Peak in % (%2Peak, the lower the better).

The Perf metric gives one important insight discussed earlier: the tree construction phase is the bottleneck in TBB and S-CPU, while the OWM traversal is the bottleneck in CUDA. However, the AP metric tells that the SYCL compiler has room for improvement for the tree construction in S-CPU, and in that ideal case the OWM traversal would be the bottleneck in both the SYCL and CUDA versions.

We also want to compare the different architectures in terms of performance vs power measured as throughput per watt (Millions of points per second and watt). This power-efficiency results are shown in Figure 3.18. Despite the limited power-efficiency resources available on the iGPU, the S-iGPU version achieves impressive efficiency values ranging from 1.11 to 1.34 Mpoints/s/W across the tested clouds. These efficiency figures surpass the best TBB CPU version, which ranges from 0.41 to 0.68 Mpoints/s/W, and are comparable to the efficiency of

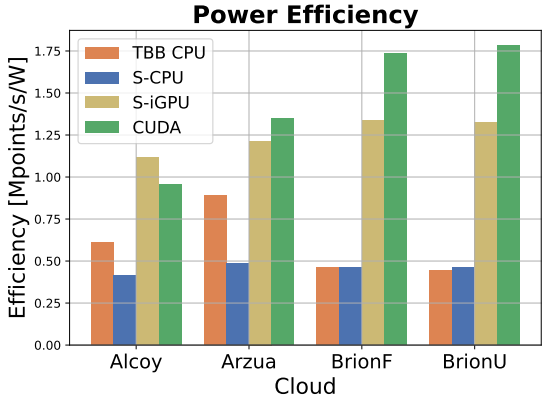



Figure 3.18: Power efficiency of the best TBB, SYCL and CUDA implementations.

the CUDA version, which ranges from 0.96 to 1.78 Mpoints/s/W. The high power efficiency of the S-iGPU version can be attributed to the low power consumption of the device, with a TDP of only 15 W, compared to the 95 W TDP of the CPU and the 160 W TDP of the dGPU. This power efficiency makes the S-iGPU version an attractive option for scenarios where energy consumption is a critical concern, such as in mobile or embedded systems, or in large-scale deployments where cumulative power savings can be significant.

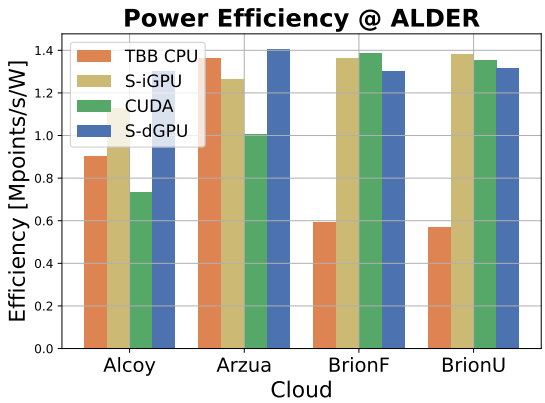
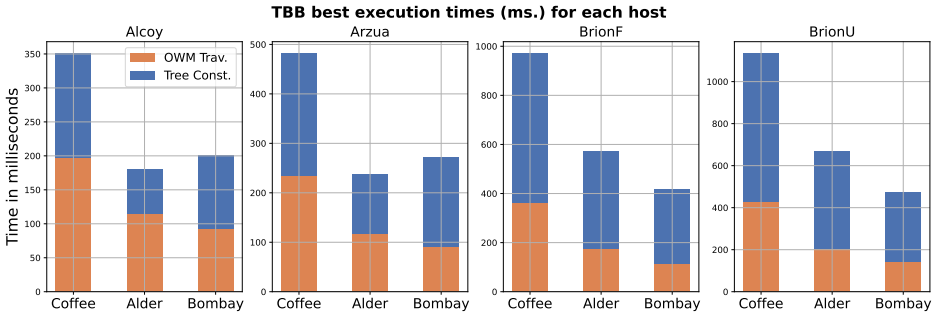


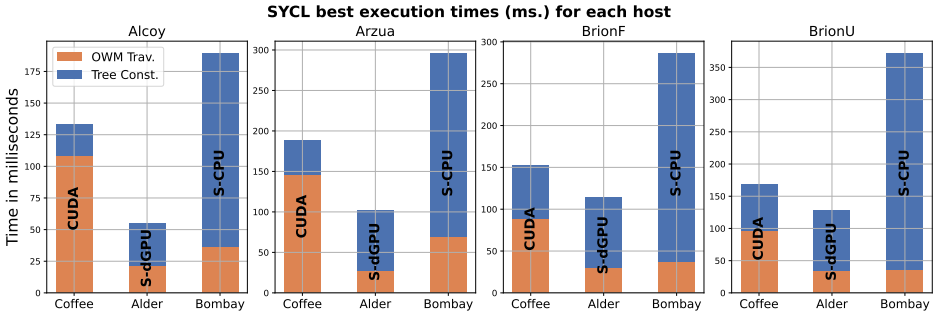

Figure 3.19: Power efficiency of the best TBB, SYCL and CUDA implementations on the `Alder` platform.

This is even more evident in the `Alder` platform, which results are shown in Figure 3.19. This results support the ideas already discussed in the Chapter 2 about the potential of new architectures aligned with the development of new programming models, such as SYCL, to achieve high computational efficiency in power-constrained environments, that are becoming increasingly important in the era of edge computing and Internet of Things (IoT).

### 3.5.4. Cross platform performance comparison




(a) Execution times for the best TBB implementation on each platform.




(b) Execution times for the best SYCL implementation on each platform: CUDA, S-dGPU, and S-CPU, for `Coffee`, `Alder`, and `Bombay`, respectively.

Figure 3.20: Cross platform performance comparison for the best TBB and SYCL implementations.

In this final section, we compare the performance of the best TBB and SYCL implementations on the three platforms described in Section 3.2.5. Figure 3.20 shows the execution times. The best TBB implementation is O4 MinRadius for the three platforms, while the best SYCL implementations are CUDA, S-dGPU,

and S-CPU, respectively:

- For the TBB implementation, the `Coffee` platform exhibits the slowest execution times, while `Alder` and `Bombay` demonstrate comparable performance. Specifically, `Alder` achieves 1.83x faster execution times than `Coffee`, aligning with the expected performance difference based on their respective CPU capabilities. Notably, the performance gap between `Alder` and `Bombay` is less pronounced, with `Bombay` being only 1.14x faster on average. Interestingly, for the Alcoy and Arzua datasets, `Bombay` actually performs slower than `Alder`. This observation corroborates our earlier analysis regarding the memory-bound nature of the algorithm, highlighting the importance of memory subsystem design in massive sensor-based data processing.
- The optimal SYCL implementations across the three platforms are CUDA, S-dGPU, and S-CPU, respectively. GPU-based executions consistently deliver the fastest performance. Particularly noteworthy is the impressive performance of the `Bombay` CPU in the OWM traversal phase, lagging only 32% behind the `Alder` GPU. However, the `Bombay` CPU's limitations become more apparent in the tree construction phase, where it is 71% slower than the `Alder` dGPU. An additional consideration for the `Bombay` CPU is its higher TDP, exceeding that of the `Alder` GPU by 65W. This significantly higher power consumption renders the `Bombay` CPU less competitive in terms of power efficiency, an important factor in the design of systems as we previously discussed in Chapter 2.

These findings underscore the complex interplay between different heterogeneous computing architectures and their impact on massive sensor data processing. They highlight the need for careful consideration of both performance and power efficiency when selecting platforms for accelerating sensor-based analytics tasks.

## 3.6. Conclusions

We have explored different optimization strategies adapted to the specific capabilities of the devices that we target in this study (CPU, iGPU and dGPU), as well as different programming environments that currently represent the state-of-the-art in heterogeneous programming (OpenMP, TBB, SYCL, CUDA) using as case study a C++ tree based application that processes LiDAR data point clouds (the OWM function).

Regarding the optimizations proposed in this chapter, our results hint that optimizations oriented to minimize memory access such as O3 (memoization) and O4 (tuning granularity) have a greater impact on the CPU, while optimizations oriented to tune the tree data structure to the device (O1) and exploit all available parallelism (O2) have a greater impact on the GPU.

Regarding the programming environments, SYCL is a promising abstraction that promotes programmers productivity by enabling the execution over different heterogeneous devices (CPU, iGPU or dGPU) with a single source code. Moreover, we have found that SYCL is highly competitive (mainly if you care about code portability) with device-specific programming environments such as TBB (CPU) or CUDA (GPU) for the type of application studied here. For future research lines we will consider hybrid implementations that use the CPU and the GPU at the same time, as well as the evaluation of the implementation in low-power devices that can be coupled with the LiDAR sensor.

# 4 Accelerating Epileptic Seizure Detection using Patterns augmented by Features

## 4.1. Motivation and summary

Epilepsy is a chronic disorder of the central nervous system affecting more than 50 million people worldwide, making it one of the most common neurological diseases globally [134]. An epileptic seizure is a sudden, transient aberration in the electrical activity of the brain that may produce a lapse in attention, sensory hallucination, whole-body convulsion [109, 110], and it significantly increases an individual's chance of experiencing burns, skull fractures, and even sudden unexpected death [87, 114]. The detection of epileptic seizures is crucial in the diagnosis and treatment of epilepsy, and electroencephalography (EEG) is a commonly used noninvasive method for this purpose, as it assesses the electrical activity produced by brain regions from the scalp surface after being captured by electrodes and conductive materials. The electrocortiogram is the EEG obtained directly from the cortical surface, whereas the electrogram is the EEG acquired with depth electrodes. Our research focuses only on the EEG recorded from the scalp surface [111]. The manual analysis of EEG data can be time-consuming and subject to human error, which can lead to diagnostic delays and suboptimal treatment. In recent years, there has been an increasing interest in

the use of automatic detection algorithms to improve the accuracy and efficiency of EEG-based seizure detection [100].

Methods for detecting seizures that do not involve EEG include the use of accelerometers, electromyography, and electrodermal activity sensors that can be worn on the wrist or arm. These solutions have been shown to be effective in detecting generalized tonic-clonic seizures (GTCS) and major motor seizures but may not be as effective in detecting other types of seizures [12, 15, 69, 79]. On the other hand, EEG-based solutions have the potential to detect a wide range of seizure types, but they can be sensitive to interference from external factors, such as electrical artifacts [10, 14, 88]. Therefore, signal preprocessing and cleaning is necessary for EEG-based epileptic seizure detection in order to improve the accuracy and reliability of EEG data analysis.

Most of the existing methods for EEG-based seizure detection rely on the use of a set of predefined features extracted from EEG signals [115]. According to different sets of features and EEG channels (pairs of electrodes), several Machine Learning (ML) models (Support Vector Machine -SVM-, k-Nearest Neighbor -kNN-, Decision trees -DT-, Random Forest -RF-, Artificial Neural Networks -ANN-, etc.) [2, 3, 17, 82, 99, 113] can be trained to classify seizures and non-seizures. However, as we will detail in the next section, these methods usually require the analysis of several EEG channels, train with 80-90% of the available data in the EEG databases (leaving little room for validation), and even so, do not achieve good enough sensitivity without false alarms [43, 115].

More recent, and much less popular are the approaches that treat the EEG signal as a time series and rely on pattern matching based on Euclidean Distance (ED) or Dynamic Time Warping (DTW) distance to compare the signal with a set of predefined discriminatory patterns (aka shapelets) [104, 117, 137]. However, there is plenty of room for improvements in this area as we can see in one of the latest works [117] where the seizure patterns have to be manually identified via visual inspection of different seizures in different channels.

To the best of our knowledge, our approach, *PaFESD*, is the first one conflating these two mentioned strategies: the use of DTW to automatically find patterns that discriminate seizures from non-seizures regions of an EEG channel combined with feature based analysis of the signal in order to filter out non-seizure regions that can mislead the pattern matching phase. We also contribute with a new quality metric that we call *Discrimination Ratio*, *DR*, that has been devised to search seizure patterns that do not produce false positives (false alarms). We demonstrate that our method can automatically find discriminative seizure patterns even when we only use around 22% of the EEG signal (at most 4 seizures)

to train our model. Besides, we also leverage signal filtering and artifact rejection techniques to reduce the noise due to muscle movements or electrical interferences. Finally, our proposal automatically detects the most discriminative channel so that, once the seizure patterns have been identified in that channel, they can be used for a seizure detection algorithm running on a wearable connected to only two electrodes (in glasses, headband, headset or similar). The codes required to reproduce our results are available in GitHub[1].

We thoroughly describe these contributions in the following sections. First, we discuss in Sec. 4.2 the related work regarding automatic seizure detection and the limitations of current methods. Next, in Sec. 4.3 we characterize the main ideas of our proposal and outline a more algorithmic view in Sec. 4.4. Finally, we present the experimental evaluation in Sec. 4.5 and conclude in Sec. 4.6.

## 4.2. Background and Related Work

EEG-based seizure detection algorithms can be broadly classified into feature and non-feature based [74]. These features are usually categorized as morphological (amplitude, zero crossings, etc.), time domain (mean, variance, kurtosis, energy, etc.), frequency domain (power spectrum, spectral entropy, etc.) and time-frequency domain (Line Length, discrete wavelets (WT), Shannon entropy, etc.) [43, 115, 138]. In particular, a wealth of previous studies combine feature extraction from the EEG signals and ML classification models such as SVM, RF, kNN, or ANN [49, 50, 115]. These strategies are able to classify seizures and non-seizure regions of the EEG. However, there is no consensus yet in the selection and combination of the best features and models, although variations of Line Length and WT combined with RF and SVM have been among the most successful ones [115].

In the non-feature based alternatives, the signal is directly analyzed by, for example, Deep Neural Networks, DNN, (Convolutional Neural Networks, CNN [26, 51], Recurrent Neural Network, RNN [101], Federated Learning [8], etc.) or with pattern matching algorithms [104, 117, 137]. DNN approaches require a significant amount of labeled training data which is not always available for personalized seizure detection. Pattern matching strategies find epileptic patterns comparing signal subsequences using time-series distances like Euclidean Distance (ED) or Dynamic Time Warping (DTW). However, little work has been published in this direction although we believe it is a promising one.

[1] https://github.com/PPMC-DAC/PaFESD-Epileptic-Seizure-Detection

In Table 4.1 we summarize some recent studies that have been validated using the CHB-MIT EEG database [38]. We highlight the number of patients, the % of data used for training, the choice of cross-validation, the features and classification techniques, and the quality metrics reported.

| [Ref] (Year) | #Pat. | % Train | Cross Val. | Features | Classifier | SEN | SPE |
|---|---|---|---|---|---|---|---|
| [99] (2016) | 24 | 100% | 10-fold | STFT | RF | 97% | 99% |
| [17] (2017) | 18 | 94% | LOO/patient | DWT | SVM | 91.7% | 92.9% |
| [3] (2018) | 22 | 90% | 10-fold | DWT/WPD | RF/SVM/kNN/ANN | 100% | 100% |
| [82] (2020) | 21 | 80% | 5-fold | DWT | DNN | 92.4% | 96.0% |
| [2] (2022) | 24 | 66% | No | Energy, DWT | SVM | 93.5% | 94.6% |
| [117] (2022) | 24 | 100% | No | No | DTW threshold | 95.5% | – |
| [84] (2015) | 3 | 100% | No | Line Length | SVM | 81.5% | 88.9% |
| [20] (2019) | 23 | 81% | LOO/seizure | No | CNN, RNN | 99.7% | 99.6% |
| **This work** | 24 | 22% | No | Amp, Energy, Band Power | DTW *DR* | 99.6% | 100% |

Table 4.1: Comparison with previous studies using CHB-MIT

Regarding these quality metrics, it is common place to measure the number of true positives, *TP*, false positives, *FP*, true negatives, *TN*, and false negatives, *FN*, and report some main metrics of the confusion matrix, as:

- **Precision (PRE)**: Also known as positive predictive value (PPV), precision is the proportion of positive cases that the model predicted correctly. It is calculated as $PRE = TP/(TP + FP)$, so it focuses on the relevance of the positives (PRE=1.0 means that there are no *FP* or false alarms, i.e., zero non-seizures detected as such), but it does not consider the *FN* (missing actual seizures).
- **Sensitivity (SEN)**: Also known as recall, hit rate, or true positive rate (TPR), SEN is the proportion of actual positive cases that the model correctly identified as such. It is calculated as $SEN = TP/(TP + FN)$, so it is equal to 1.0 when we do not miss any seizure ($FN$=0), but it does not consider the *FP* (false alarms).
- **$F_1$ score**: It is a measure of a classifier accuracy that considers both PRE and SEN. It is the harmonic mean of PRE and SEN, giving equal weight to both, $F_1 = 2(PRE \cdot SEN)/(PRE + SEN)$. When $F_1 = 100\%$ there are no false alarms nor missed true seizures.

Many related works [115] report the accuracy (**ACC**) as a quality metric too. However, the accuracy is a misleading metric for imbalanced datasets as it is the one at hand, where the number of non-seizure events is much larger than the number of seizure ones. This is because **ACC**= $(TP + TN)/(TP + TN + FP + FN)$ and with $TN >> (TP + FP + FN)$ the accuracy easily tends to 100%. A similar issue arises using the specificity, **SPE**= $TN/(TN + FP)$. Therefore, in our evaluation we use the $F_1$ score as a more reliable quality metric, although in

Table 4.1 we also report SEN, and SPE for comparison purposes.

In the revision of the related works we have identified some practices that we would rather avoid. In addition to the use of sometimes misleading quality metrics (as ACC and SPE in unbalanced problems), we have found several works that do not report results for the whole database, DB. For example, although the CHB-MIT DB includes 24 cases and 980 hours of EEG recordings, in [84] only 3 patients and 15 hours are studied, in [17] only 18 patients are considered, and in [82] they removed 3 patients. Regarding [3] we would also report perfect results if we exclude four patients out of our study (even training with around 22% –instead of 90%– of the available data).

In some studies the unbalanced problem is converted into a balanced one for the sake of equalizing the representativity of seizures and non-seizures. Out of the 980h in the CHB-MIT DB, in [17] only 38h with seizure and 38h with no-seizure are analyzed (just 8% of the DB). In [3] only 2.2h per class (0.4% of the DB), 3.3h per class (0.7%) in [2] and in [84] 15h (1.5%). Transforming a real unbalanced problem into a hypothetical balanced one (in which *FP* can be avoided because most non-seizure data is disregarded) is not in our plans. We process the whole DB, and we report very low *FP* rates after analyzing the full extent of the non-seizure signal.

Besides, the validation strategy is in many cases also artificial. It is true that the available recorded and labeled seizure data is scarce. However, the usual workaround based on Leave-One-Out Cross Validation (LOOCV) or n-fold cross validation is not convincing in our opinion. These strategies are based on dividing the data into n folds, train with n-1 folds and validate with the remaining one, which is equivalent to train with $(n-1)/n \cdot 100\%$ of the data. This is exercised using different folding granularities: in [17] they use LOOCV at a patient granularity (train with 17 patients and validate with the remaining one) and in [20] it is used at a seizure level (train with all seizures but one). We strive to solve a more challenging problem in which we validate with a large amount of unseen data a model that is trained with scarce known signal.

The main drawback of not employing all the data for validation is that the model is not being evaluated against the most challenging data: interictal oscillations. These oscillations, despite not being seizures, exhibit similar morphology. Their nature is well-documented, proving that they are not mere random noise but, rather a complex phenomenon that can be characterized by a specific set of features [80]. Although these interictal oscillations reflect the electrical "irritability" of the brain and have clinical significance, they do not manifest the physical symptoms that commonly arise during an epileptic seizure [5, 86]. Categorize

these phenomena is beyond the purview of this thesis, and we will refer to them as interictal epochs, as will be elaborated in Sec. 4.3.1, without distinguishing between interictal epochs of different nature. Gaining inside into the properties of these abnormalities would indeed be a crucial step towards the development of a seizure detection algorithm.

To that end we explore the pattern matching alternative. It was also used in [117] obtaining a mean sensitivity of 95.5% across all 24 patients in the CHB-MIT database. However, this approach requires a visual identification of the seizure pattern(s), does not adequately address the crucial aspect of how different patterns are combined for each patient and uses the entire signal recording for training. Inspired by this work, our approach augments the seizure patterns detection based on DTW by combining classification and signal feature extraction to help in removing non-seizure regions out of the analysis. Our method automatically identify the discriminative channels and patterns, it uses preprocessing methods to clean the signal from noise and artifacts, and achieves perfect $F_1$ score for almost all patients, even under scarce training data.

## 4.3. Proposal overview and main ideas

This section presents a formal description of the seizure detection problem and outlines the primary components of our proposed solution. We begin by establishing the key concepts and notation utilized throughout this chapter, providing a foundation for subsequent discussions. Following this, we offer a comprehensive overview of the essential steps comprising our approach.

### 4.3.1. Key Concepts

We rely on Figure 4.1 to illustrate key concepts that are required to understand the EEG analysis we propose, and we introduce the notation and terminology used throughout this section.

**Definition 4.3.1 (Time Series)** A multi-channel time series $T^p$ for a patient $p$ is a collection of signals, one per channel. It can be represented as $T^p = \{S^{c_j};\ 1 \leq j \leq nc_p\}$ where $S^{c_j}$ is a discrete-time signal or channel, $nc_p$ is the number of channels for patient $p$, and $c_j$ is the channel label. Thus, $T^p$ can be represented as a matrix of size $n \times nc_p$, where $n$ is the length of the time series. When it is not necessary to specify a particular patient, we simply omit $p$ and

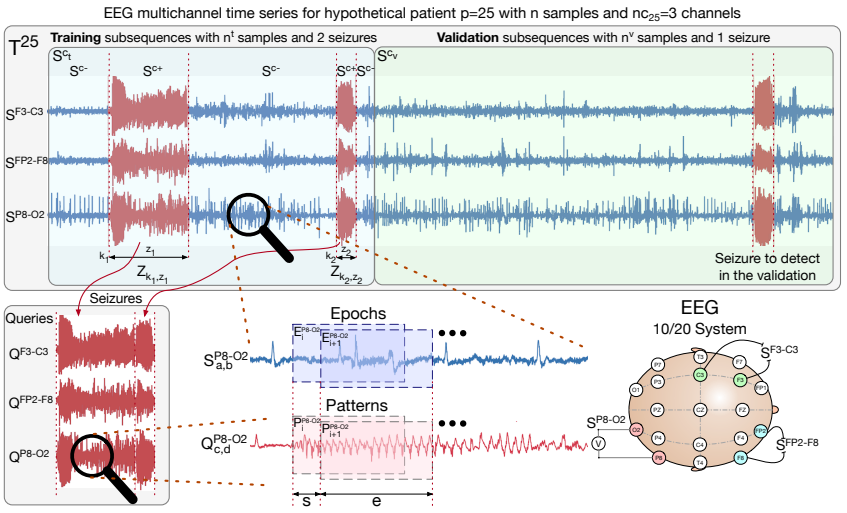



Figure 4.1: Main concepts and notation used throughout this chapter.

use $T$ for a general time series. In Figure 4.1, a hypothetical $T^{25}$ and its $nc_{25} = 3$ channels are shown.

As defined, $T$ can also be seen as a multivariate time series, where each sample $t_i$ is a vector of $nc_p$ real numbers. However, our research focuses on the analysis of each channel independently, i.e., we will treat each channel as a univariate time series. This will enable the detection of the most discriminative channel for each patient, which will be used to build a wearable detection device with only two electrodes, a requirement for our future work.

**Definition 4.3.2 (Signal/Channel)** A signal or channel, $S^c$, is a discrete-time sequence of real-valued numbers $s_i^c \in \mathbb{R}$, i.e., $S^c = \{s_i^c;\ 0 \leq i < n\}$, where $n$ is the length of the sequence and $c$ is a channel id or label. Each number $s_i^c$ represents the electrical potential between two electrodes sampled at a specific time $t_i$. In the CHB-MIT database the sampling frequency is 256 samples per second, sps, i.e. Fz=256Hz.

Channel labels in the 10-20 system [25] are used to identify each channel. For example, the channel F3-C3 represents the electrical potential measured between electrodes F3 and C3 and will be identified as $S^{F3-C3}$. Throughout this research, we will use the terms signal and channel interchangeably. In Figure 4.1 we show the channels $S^{F3-C3}$, $S^{FP2-F8}$ and $S^{P8-O2}$ and the electrode labels of the 10/20

system.

In the used EEG database there are different recordings of each patient. Therefore, we build $T^p$ by concatenating all the recordings from a single patient, $p$. For example, if for a patient $p$ the database includes 10 recordings of 1 hour each, and each recording has 23 channels, then $T^p$ can be represented as a matrix of $9,216,000 \times 23$ samples, where $n = 9,216,000$ and $nc_p = 23$.

**Definition 4.3.3 (Subsequence)** A subsequence is a subperiod in a single channel. For a channel $S^c$, the subsequence $S^c_{k,l}$ is a continuous subset of values such that $S^c_{k,l} = \{s^c_i;\ k \leq i < k + l\}$, where $k$ is the starting index of the subsequence, $l$ is its length and $s^c_i$ are the samples comprised by the subsequence. Subsequence variables only store the starting index $k$ and length $l$, so they do not duplicate the represented samples.

**Definition 4.3.4 (Seizure)** A seizure, denoted by $Z^c_{k,z}$, is a subsequence in channel $S^c$ that starts at the sample $k$ and has a length $z$, i.e. $Z^c_{k,z} = S^c_{k,z}$ when this subsequence has been labeled as a seizure. In the dataset used, ictal (seizure) and interictal (non-seizure) episodes are clearly identified through metadata, $md$, that specifies the onset and offset timestamps of each seizure, from which we gather the $k$ and $z$ values. Each seizure belongs to a single seizure episode and all the channels in a time series $T$ share the same timestamps. More precisely:

1. The number of seizures in $T$, denoted by $N_z$, varies between patients. In the dataset used, it ranges from 3 to 40 seizures per patient. In Figure 4.1, there are $N_z = 3$ seizures, colored in red.
2. The presence of seizures in $T$ segments the time series into ictal, $S^{c+}$, and interictal, $S^{c-}$, subsequences for each channel $S^c$ of $T$, as we see in Figure 4.1.
3. Note that each seizure in $T$ can have a different duration, $z$. In the CHB-MIT dataset, seizure duration ranges from 9 to 264 seconds, with a mean duration of 65 seconds (16640 samples at a sampling rate of 256 sps).
4. The metadata information read from the EEG database includes the starting index, $k_i$, and length, $z_i$, of each seizure $i$, as the set $md = \{(k_i, z_i);\ 1 \leq i \leq N_z\}$.

As it has been said, our goal is to automatically detect seizure episodes in patient's EEG signals. To that end, we aim at finding one or several representative patterns in some ictal subsequence, $S^{c+}$, that can later be used to detect future ictal episodes. This can be implemented with our EEG database by partitioning $S^c$ in a training subsequence, $S^{c_t}$ with length $n^t$, and a validation subsequence,

$S^{c_v}$ with length $n^v$. $S^{c_t}$ is used to find the representative pattern or patterns during the training phase, and later, at the validation phase we assess whether the found patterns can be used to successfully identify the seizures in $S^{c_v}$ minimizing the number of false alarms (false positives). With this, we say that $N_z^t$ and $N_z^v$ are the number of seizures used for training and validation, respectively. In our experiments and for all patients, our approach ensures that $S^{c_t}$ comprises at least 20% of the signal containing between 2 and 4 seizures. If 20% of the signal has less than 2 seizures, then the number of samples in $S^{c_t}$ is increased to at most 50% of the total signal while maintaining $2 \leq N_z^t \leq 4$. This strategy guarantees a representative subset of seizure data for training without manual intervention. By doing so, we ensure that our method can generalize well to unseen data and effectively capture representative patterns. In our example of Figure 4.1, $N_z^t = 2$ and $N_z^v = 1$.

From now on, we refer to the training subsequence of the time series, unless otherwise stated, since this is the one we automatically analyze to find representative seizure patterns. The validation subsequence will be used in the experimental section to assess the PRE, SEN and $F_1$ score metrics achieved by our proposal. Anyway, more definitions are required in order to understand the process of automatically identifying seizure patterns, which we introduce next.

**Definition 4.3.5 (Query)** A query on channel $S^c$, $Q^c$, is the concatenation of all $N_z^t$ seizures in $S^{c_t}$ one after the other in the order they appear. The concatenation of subsequences can be defined as the operation of appending one subsequence to another. This can be expressed as: $Q^c = (Z^c_{k_1,z_1}, Z^c_{k_2,z_2}, ..., Z^c_{k_{N_z^t},z_{N_z^t}})$. The total length of the query is $n_q = \sum_{i=1}^{N_z^t} z_i$. For example, in Figure 4.1 we see that $Q^{F3-C3}$ is the concatenation of the two training seizures in the channel F3-C3. A subsequence of the query $Q^c$ starting at index $k$ and length $l$ is identified as $Q^c_{k,l} = \{q_i^c;\ k \leq i < k + l\}$ being $q_i^c$ the samples of the query.

Our approach, combines discrete-time signal processing based on feature extraction (in order to filter out non-seizure regions) and time series data analytics (for the seizure pattern discovery). These two analyses are carried out on the individual channels, $S^c$, of the time series, $T$. That way and depending on the particular patient, there may be channels that include more preserved seizure patterns than others. By working at the channel level, for all the channels in the time series, we can identify these relevant channels. Considering that our EEG database usually includes 23 channels per patient and several millions of samples per channel, for the pattern discovery phase it is clearly prohibitive to analyze all possible combinations of subsequences $S^{c_t}$ and queries $Q^c$ at all possible start-

ing positions and lengths. As a workaround, we will only analyze fixed-size and equidistant chunks of $S^{c_t}$ (called *epochs*) and $Q^c$ (called *patterns*), as we define next.

**Definition 4.3.6 (Epoch)** In order to find seizure patterns in a channel $S^c$, we only consider subsequences with a fixed length, $e$, and a fixed stride, $s$. We call epochs to these particular subsequences that, in other words, virtually segment the channels into smaller equidistant and fixed-size sliding windows. More precisely, an epoch $E_i^c = S_{k,e}^c$ with $k \in \{i \cdot s;\ 0 \leq i < n_e\}$, being the number of epochs $n_e = \lfloor (n-e)/s \rfloor + 1$. In plain words, the $i$-th epoch, $E_i^c$ of a channel $S^c$, is the subsequence $S_{k,e}^c$ with a fixed length $e$ that starts at the index $k = i \cdot s$, being $s$ the epoch stride. In Figure 4.1, we depict just two generic epochs of the subsequence $S_{a,b}^{P8-02}$, labeled as $E_i^{P8-02}$ and $E_{i+1}^{P8-02}$.

Note that only full $e$ length epochs are considered and therefore the number of epochs in a channel of length $n$ is $n_e = \lfloor (n-e)/s \rfloor + 1$. For example if $S^c$ has $n = 11$ samples, the epoch size is $e = 4$ samples and the stride is $s = 2$ samples, we end up with $\lfloor (11-4)/2 \rfloor + 1 = 4$ epochs $E_i^c$ with indices $i = \{0, 1, 2, 3\}$ and starting at positions $k = \{0, 2, 4, 6\}$. $E_4^c = S_{8,4}^c$ is not considered because it overflows the $n = 11$ size.

The length, $e$, and stride, $s$, are constant input parameters of our application that can only be modified at different runs. Special care is taken so that there are no epochs straddling $S^{c+}$ and $S^{c-}$ borders (ictal and interictal borders). This translates in that we have epochs windowing $S^{c+}$ and epochs doing the same in $S^{c-}$, that will be called $E^{c+}$ and $E^{c-}$, respectively.

**Definition 4.3.7 (Pattern)** Similarly to the epochs, the patterns are the subsequences or sliding windows of size $e$ and stride $s$ that fill in a query $Q^c$. This is, the pattern $P_i^c = Q_{k,e}^c$ with $k \in \{i \cdot s;\ 0 \leq i < n_p\}$, being $n_p = \lfloor (n_q - e)/s \rfloor + 1$ the number of patterns in the query of length $n_q$. All the patterns in $Q^c$ will be analyzed during the training phase in order to find the most discriminative ones. In Figure 4.1, we see the patterns $P_i^{P8-02}$ and $P_{i+1}^{P8-02}$ of a query subsequence $Q_{c,d}^{P8-02}$.

**Definition 4.3.8 (Batch)** A batch $B^c$ is the main output of the training phase and comprises the patterns $P_i^c$ that are identified during the training as the ones exhibiting conservative features that best discriminate seizures from non-seizures. The number of found patterns is the size of the batch, $n_b$. With this, $B^c = \{P_i^c;\ 1 \leq i \leq n_b\}$. Our training method automatically finds a batch for each patient, that is later used during the validation phase to identify the seizures in the corresponding validation subsequences $S^{c_v}$.

The process to compute the batch is explained in the following subsections, but before that, let us give more details about the EEG database used in this research.

#### 4.3.1.1. Input Dataset

For the purpose of this study, we used the CHB-MIT database [38]. The data was collected from 22 subjects (5 males and 17 females). The recordings are grouped into 24 cases, with each case stored in a number of files (from 9 to 42) from a single subject. The signals were collected at 256 samples per second with a 16-bit resolution. The International 10-20 system of EEG electrode in bipolar montage and nomenclature [25] is used.

Table 4.2 shows the relevant information of each case: patient id, number of channels ($nc_p$), total number of samples in millions ($n$), number of samples of seizure subsequences ($n^+$) and non-seizure ones in millions ($n^-$), and number of seizures ($N_z$). We also include details of our experimental setting, as the number of millions of samples used for training ($n^t$) and validation ($n^v$), number of seizures used for training ($N_z^t$) and number of samples of the query ($n_q$).

### 4.3.2. Pattern-Epoch distance and Discrimination Ratio

Let us remember that our goal is to automatically identify one or several patterns that can discriminate between seizures and non-seizures in an EEG channel. Such suitable pattern should include a well conserved shape that is present in seizures epochs, $E^+$, but not in non-seizures ones, $E^-$. This can be achieved via comparing patterns and epochs by computing the distance between them in order to measure their similarities. In some previous works tackling the EEG epileptic seizure detection problem [104], the pattern selection criterion has been based on minimizing the distance between the pattern and all trained seizure epochs, $E^+$ (i.e. on finding a nearest neighbor, NN). However, the EEG signals are very noisy and in many cases the seizures do not exhibit a well conserved pattern. We have found that for this problem a good pattern is better selected by minimizing the distance to seizure epochs, $E^+$, whilst maximizing the distance to the non-seizure epochs $E^-$. The metric we use to identify these discriminative patterns is what we call *discrimination ratio* (*DR*), that we define next.

But first, let us define the distance, $d(P_i^c, E_j^c)$, between a pattern, $P_i^c$ and an epoch, $E_j^c$, as the Dynamic Time Warping (DTW) [106, 107] distance between the two subsequences. DTW is an increasingly used algorithm for measuring

| p | $nc_p$ | $n$ [M] | $n^+$ | $n^-$[M] | $n^t$[M] | $n^v$[M] | $N_z$ | $N_z^t$ | $n_q$ | hours |
|---|---|---|---|---|---|---|---|---|---|---|
| 01 | 23 | 37.37 | 113152 | 37.26 | 7.54 | 29.84 | 7 | 2 | 17152 | 40.55 |
| 02 | 23 | 32.50 | 44032 | 32.46 | 6.56 | 25.94 | 3 | 2 | 23040 | 35.27 |
| 03 | 23 | 35.02 | 102912 | 34.92 | 7.07 | 27.96 | 7 | 2 | 30976 | 38.00 |
| 04 | 23 | 143.83 | 96768 | 143.73 | 28.83 | 115.00 | 4 | 2 | 40960 | 156.06 |
| 05 | 23 | 35.94 | 142848 | 35.80 | 7.25 | 28.70 | 5 | 2 | 57600 | 39.00 |
| 06 | 23 | 61.50 | 39168 | 61.46 | 12.36 | 49.14 | 10 | 2 | 6400 | 66.73 |
| 07 | 23 | 61.80 | 83200 | 61.71 | 12.42 | 49.38 | 3 | 2 | 46592 | 67.05 |
| 08 | 23 | 18.44 | 235264 | 18.20 | 3.75 | 14.69 | 5 | 2 | 82944 | 20.01 |
| 09 | 23 | 62.55 | 70656 | 62.48 | 12.57 | 49.98 | 4 | 2 | 36608 | 67.87 |
| 10 | 23 | 46.10 | 114432 | 45.99 | 9.28 | 36.82 | 7 | 2 | 31488 | 50.02 |
| 11 | 23 | 32.07 | 206336 | 31.86 | 6.47 | 25.59 | 3 | 2 | 13824 | 34.79 |
| 12 | 23 | 21.84 | 377600 | 21.46 | 4.41 | 17.43 | 40 | 4 | 65024 | 23.69 |
| 13 | 23 | 30.41 | 157440 | 30.26 | 9.18 | 21.23 | 12 | 4 | 48896 | 33.00 |
| 14 | 23 | 23.96 | 43264 | 23.92 | 4.85 | 19.11 | 8 | 2 | 10752 | 26.00 |
| 15 | 26 | 36.87 | 509952 | 36.36 | 7.44 | 29.44 | 20 | 4 | 124160 | 40.01 |
| 16 | 23 | 17.51 | 79872 | 17.43 | 8.81 | 8.70 | 10 | 4 | 39936 | 19.00 |
| 17 | 23 | 18.81 | 75008 | 18.73 | 3.79 | 15.01 | 3 | 2 | 52480 | 20.41 |
| 18 | 18 | 32.84 | 81152 | 32.76 | 9.91 | 22.93 | 6 | 2 | 20480 | 35.63 |
| 19 | 18 | 27.58 | 60416 | 27.52 | 5.58 | 22.01 | 3 | 2 | 39680 | 29.93 |
| 20 | 28 | 25.44 | 75264 | 25.36 | 7.69 | 17.75 | 8 | 2 | 21504 | 27.60 |
| 21 | 23 | 30.26 | 50944 | 30.21 | 6.11 | 24.14 | 4 | 2 | 27136 | 32.83 |
| 22 | 23 | 28.57 | 52224 | 28.52 | 5.78 | 22.80 | 3 | 2 | 33792 | 31.00 |
| 23 | 23 | 24.48 | 108544 | 24.37 | 6.18 | 18.30 | 7 | 2 | 28416 | 26.56 |
| 24 | 23 | 19.63 | 137216 | 19.49 | 3.98 | 15.64 | 20 | 4 | 29696 | 21.30 |
| **Mean** | 22 | 37.72 | 127402 | 37.59 | 8.24 | 29.48 | 8 | 2 | 38730 | 40.93 |

Table 4.2: Details of the CHB-MIT EEG database used in this study

similarity between two temporal sequences that may vary in phase or speed. Originally used for automatic speech recognition (to cope with different speaking speeds), it is now recognized as one of the most reliable similarity metrics [22]. Its higher computational cost in comparison with cheaper distances (as the Euclidean Distance, ED), has spurred the development of many simplified and optimized variants. One of the first simplifications was the cDTW (constrained DTW) [106] that limits the warping path to a band, known as the Sakoe-Chiba band or warping window, $w$, around the main diagonal of the cost matrix. When the warping window is set to 0, the cDTW degenerates into the ED. The cDTW is a good compromise between the ED and the DTW, as it is faster than the DTW but more accurate than the ED. Our research uses cDTW with warping window, $w = 16$, as the distance metric between patterns and epochs. When the cDTW is used to find a nearest neighbor, NN, for example to find the most similar epoch to a pattern, further optimizations have been proposed, as lower-bounding, early abandoning, and pruning techniques [44].

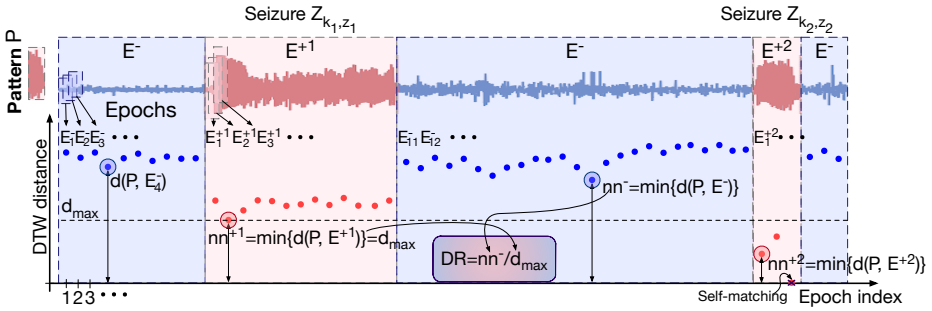



Figure 4.2: Distances between one pattern, $P$, and all epochs required to calculate the discrimination ratio, $DR$.

We rely on Figure 4.2 to exemplify the main idea under our discrimination ratio, $DR$, metric. In the figure we depict a generic pattern, $P$, from the query $Q^{F3-C3}$ and all the epochs, $E_j$, in the training subsequence $S^{F3-C3_t}$. We identify with $E^- = \{E_j^-\}$ the set of non-seizure epochs and with $E^{+z} = \{E_i^{+z}\}$ the set of epochs in each seizure, $z$. For example, in Figure 4.2 we have two seizures and the corresponding sets of seizure epochs are $E^{+1}$ and $E^{+2}$. Each blue point in the figure represents the DTW distance between the pattern, $P$, and one epoch, $E_j^-$ (e.g. $d(P, E_4^-)$), whereas the red points represent the distances between the pattern, $P$, and the seizure epochs, $E^{+z}$.

Now, we define the pattern $P$'s nearest non-seizure neighbor, $nn^- = min\{d(P, E^-)\}$, that is the most similar non-seizure epoch. This is the worst case scenario for

a good discriminative pattern, which we would like to be as different (far away) as possible to all non-seizure epochs. Similarly, we define the pattern $P$'s nearest seizure neighbor, $nn^{+z} = min\{d(P, E^{+z})\}$, that is the most similar seizure epoch for each seizure, $z$. For example, in Figure 4.2 we have $nn^{+1}$ and $nn^{+2}$. Note that since $P$ is a pattern from the query (that was built out of concatenating all training seizures), there will be a self-matching or trivial match that results in a distance $d(P, P) = 0$. We discard this trivial match as we see in the figure for the distance $d(P, E_3^{+2}) = 0$. In order to compute the quality of our pattern $P$ we consider again the worst case scenario and define $d_{max}$ as the furthest nearest neighbor of all seizures ($nn^{+1}$ in the figure). More precisely, $d_{max} = max\{nn^{+i};\ 1 \leq i \leq N_z^t\}$.

With all that, we can finally define the discrimination ratio, $DR = nn^-/d_{max}$, as the ratio between the nearest non-seizure neighbor, $nn^-$, and the furthest nearest seizure neighbor, $d_{max}$. We say that $P$ is a good discriminative pattern if the associated $DR$ is greater than one, meaning that using $d_{max}$ as a discrimination threshold we can detect all the seizures without false positives. For example, if in Figure 4.2, $nn^- < d_{max} \rightarrow DR < 1$, and if we use $d_{max}$ as the discrimination threshold, then at least one non-seizure epoch would be falsely detected as a seizure (false positive).

### 4.3.3. Distance matrix and discrimination vector

As we have just said, the discrimination ratio, $DR$, quantitatively measures the quality of a pattern, $P$. The larger $DR$, the more discriminative is the pattern. However, there are many patterns in the query, so we need to compute the discrimination ratio for all of them in order to find the best one. To do so, we first compute the distance matrix, $DM$, between all patterns, $P_i$, and all epochs, $E_j$, as shown in Figure 4.3. Formally, we use $d_{i,j} = d(P_i, E_j^-)$ for the DTW distances between the patterns and the non-seizure epochs. Similarly, for each seizure, $z$, we compute $d_{i,j}^{+z} = d(P_i, E_j^{+z})$. Then, we compute the discrimination vector, $DV$, that contains $DR_i$ for each pattern, $P_i$, as the ratio between the nearest non-seizure neighbor, $nn_i^- = min\{d_{i,j}\}$, and the furthest nearest seizure neighbor, $d_{max,i} = max_z\{min_j\{d_{i,j}^{+z}\}\}$, as we have just explained in the previous section. We also store $d_{max,i}$ in the discrimination vector, to serve as the threshold that distinguishes seizures from non-seizures. Finally, from the discrimination vector we select the pattern, $P_m$, with the highest discrimination ratio, $m = \arg\max_i DR_i$, as the best discriminative pattern.

Two caveats should be noted here. First, the distance matrix does not need

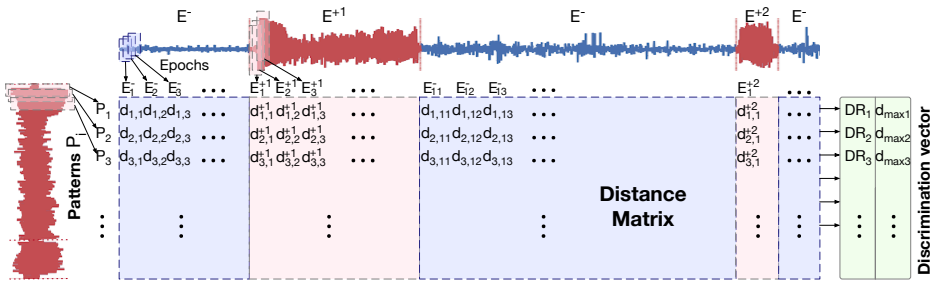



Figure 4.3: Distance Matrix and Discrimination Vector.

to be stored in memory since each entry in the discrimination vector can be computed on the fly on a row by row basis. Actually, the discrimination vector does not need to be stored either because we only have to keep the maximum $DR$, its corresponding $d_{max}$ and the id of the corresponding pattern. In other words, the distance matrix and discrimination vector are just logical concepts that give us an idea of the overall computational load.

Second, it is possible to avoid redundant computations in the distance matrix. For example, in Figure 4.3 we see that the patterns $P_i$ are just the epochs in $E^{+1}$ and $E^{+2}$ since the query is just the concatenation of the seizure subsequences. In other words, the AB-join of the patterns and the $E^+$ epochs is equivalent to the self-join of the patterns ($\{P_i\} \times \{P_i\}$). Clearly, $d(P_i, P_i) = 0$ is the trivial self-match that can be discarded. Besides, $d(P_i, P_j) = d(P_j, P_i)$. These redundancies can be avoided by computing only the upper triangular submatrix corresponding to $\{P_i\} \times \{P_i\}$ space. This is, $d(P_i, P_j)$ is computed only for $j > i$, but used for both the computation of $DR_i$ and $DR_j$ (via $d_{max,i}$ and $d_{max,j}$).

### 4.3.4. Improving $DR$: build a batch of patterns

In many cases, a single pattern is not enough to detect all seizures without false positives. For instance, consider the example depicted in Fig. 4.4, where we have two patterns, $P_1$ and $P_2$. None of them is able to detect all seizures without false positives, since $DR_1 < 1$ and $DR_2 < 1$. In the figure we can see 2 false positives for $P_1$ and one for $P_2$. However, note that $P_1$ detects very well the first seizure with a small distance $nn_1^{+1}$, but the second seizure has a larger distance $nn_1^{+2}$ that sets a $d_{max,1}$ that is above the nearest non-seizure neighbor, $nn_1^-$, which results in a $DR_1 < 1$. On the other hand, $P_2$ detects with a small

distance $nn_2^{+2}$ the second seizure, but the first one sets a $d_{max,2} = nn_2^{+1}$ larger than $nn_2^-$, resulting again in $DR_2 < 1$.

Fortunately, we can get the best of both patterns by combining them in a batch, $B^c = \{P_1, P_2\}$, that achieves a better discrimination ratio, $DR_{12} > 1$. This is, the combination of patterns can be more discriminative than each individual one. The idea is to keep the minimum nearest neighbor from each pattern, i.e. $nn_{12}^- = \min\{nn_1^-, nn_2^-\}$ and $nn_{12}^{+j} = \min\{nn_1^{+j}, nn_2^{+j}\}$; $j = 1, 2$. When the two patterns complement each other well, $d_{max}$ can decrease more than what $nn^-$ does and in many cases we obtain a discrimination ratio of the batch, $DR_{12} = nn_{12}^-/d_{max,12} > 1$. This is the case in the bottom part of Fig. 4.4, where we end up with small nearest neighbors for the seizures and a resulting $d_{max,12}$ smaller than the smallest non-seizure $nn^-$.

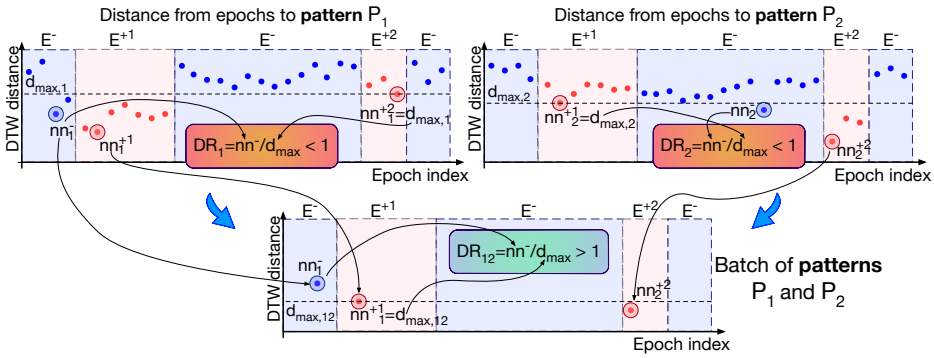



Figure 4.4: Example to illustrate the construction of a batch of two patterns.

In general, the size of the batch, $n_b$, can be larger than two, but large batches also pose some drawbacks. First, finding combinations of $n_b$ complementary patterns out of thousands of available patterns is computationally costly. Second, the larger $n_b$, the more likely it is that appear a smaller $nn^-$ which degrades the $DR$ relation. Third, the validation phase is also more expensive since we have to compute the DTW distance between the epoch (that can be a seizure or not) and all the $n_b$ patterns in the batch, and compare the minimum distance with the batch $d_{max,c}$ to take a decision. For these reasons, we have decided to limit the batch size to up to three patterns, $n_b \leq 3$, as we discuss in Sec. 6.

The general equations required to build a batch, $B^c$, of $n_b$ patterns are the following. Given the set of non-seizure nearest neighbors, $nn_i^-$, for each pattern, $P_i$, with $1 \leq i \leq n_b$, and the set of seizure nearest neighbors, $nn_i^{+z}$, for each seizure, $z$, with $1 \leq z \leq N_z^t$, and $1 \leq i \leq n_b$, we define the combined $d_{max,c}$ of

the batch as $d_{max,c} = max_z\{min_i\{nn_i^{+z}\}\}$. Similarly, we define the combined nearest non-seizure neighbor, $nn_c^- = min_i\{nn_i^-\}$. Finally, the discrimination ratio of the batch is $DR_c = nn_c^-/d_{max,c}$.

### 4.3.5. Pruning the search space using signal features

In principle, the number of DTW distances that we have to compute is equal to the number of patterns times the number of epochs. Remember that the number of patterns is equal to $n_p = \lfloor (n_q - e)/s \rfloor + 1$, being $n_q$, $e$ and $s$ the number of samples of the query, the pattern and the stride, respectively. Similarly, the number of epochs is $n_e = \lfloor (n^t - e)/s \rfloor + 1$, with $n^t$ equal to the number of samples in the training subsequence of the channel. Therefore, the number of DTW distances that we have to compute, $n_p \times n_e$, can be prohibitive. For example, by looking at the fourth patient of Table 4.2 we can easily see that for each of the 23 channels of this patient, with $e = 1280$ (5 seconds) and $s = 256$ (1 second), there are more than 112K epochs and 156 patterns, which results in more than 17.5 million DTW distances that have to be computed. Running a widely available Python DTW library[2] [37] on an off-the-shelf laptop, a single DTW distance of two subsequences of 1280 samples can take tens of milliseconds which would translate in almost a week to compute all the distances of a single channel of a single patient.

In order to speed up the computation and improve the quality metrics of interest of the algorithm, we can reduce the number of DTW distances that we have to compute and at the same time remove from the analysis the regions of the signal that are not relevant for the detection of the seizures or that are contaminated by noise or artifacts. To do so, we rely on signal processing techniques that have been used in the past to analyze EEG data. In particular, we draw inspiration from [49] and [50] where some signal features exhibited by 2-second epochs were used to try to identify epileptic seizures. These features are the amplitude of the signal, the energy of the epoch and the integral of the epoch spectral power in two different frequency bands (2.5-12Hz and 12-18Hz), which depending on thresholds set manually or adaptively can identify seizure epochs. Also, on [94] the authors propose additional EEG frequency domain analysis in order to assess the quality of the signal, in this case to validate human commands in a EEG-based brain-computer interface.

With this in mind, our proposal is to also consider physical characteristics of the signal in addition to the shape of the epochs (via DTW distance) to enhance

[2]See `https://dynamictimewarping.github.io/`

the quality of the seizure detection and reduce the computational cost. The idea is to augment each epoch with five physical attributes or features. Then, the epochs for which these features fall out of some limits are filtered out and not considered in the computation of the *DR*. These limits are automatically determined for each patient and channel, as we explain in the next section.

More precisely, for each epoch $E_i$ (and similarly for each pattern $P_i$) we compute five features[3], $f_j(E_i); 1 \leq j \leq 5$:

- $f_1(E_i)$: the peak-to-peak amplitude (pk-pk) of the epoch, that is the difference between the maximum and minimum values.
- $f_2(E_i)$: the energy of the epoch as $1/e \sum_{j=0}^{e-1} (s_j - \mu)^2$, where $e$ is the number of samples, $s_j$, in the epoch and $\mu = 1/e \sum s_j$ is the average.
- $f_3(E_i)$, $f_4(E_i)$ and $f_5(E_i)$: the band power or integrated power of the epoch in the bands 2.5-12Hz (almost covering $\delta$, $\theta$ and $\alpha$ brain waves), 12-18Hz (low band of $\beta$ waves) and 18-35Hz (high band of $\beta$ waves), respectively. This is computed as the integral in each band of the power spectral density of the epoch obtained with the Welch's method.

Now, each feature, $f_j$, is also characterized by its bounds, $b_j = \{b_j^U, b_j^L\}$, that are the upper and lower limits of the feature, respectively. With this we can compute for each feature $f_j$ and all epochs $E_i$ a do-not-compute (DNC) vector, $DNC_j$, as a binary vector of length $n_e$ (the number of epochs), that flags the epochs that fall out of the bounds. For example, if $f_1(E_i) < b_1^L$ then $DNC_1[i] = 1$ and if $f_3(E_k) > b_3^U$ then $DNC_3[k] = 1$, meaning that $E_i$ and $E_k$ should not be considered when computing the *DR*.

Figure 4.5 illustrates a hypothetical DNC vector computed for a given feature, $f_1$, and several epochs. In the chart, each blue point is a feature value (epoch amplitude in this example) within the bound and therefore the corresponding epoch is a valid one, which is identified in the DNC with a 0. However, out of bounds features (crossed in red) are flagged in the DNC with a 1.

In our proposal, epochs with at least one feature not falling within its respective bounds are discarded. This translates into a summary $DNC_S$ vector for the signal that is computed as the logical AND of all the $DNC_j$, $DNC_S = \bigwedge_{j=1}^{5} DNC_j$. Similarly, a $DNC_Q$ vector (in this case of size $n_p$) is computed for the query in order to identify the patterns $P_i$ that should not be considered during our analysis. Effectively, the ones in $DNC_S$ and $DNC_Q$ remove columns and rows, respectively, of the distance matrix.

[3]Although these five features are the ones validated in the experimental section, the method can easily accommodate others, that would just require additional experimental validation.

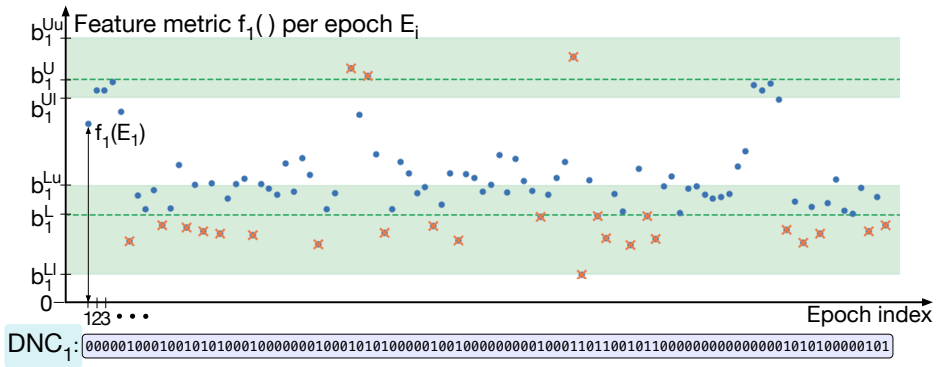



Figure 4.5: *DNC* vector for the epochs of a channel using the feature metric $f_1()$ (epoch amplitude).

The histogram's precision is a critical aspect of our analysis. It represents the distribution of our bounds, and its accuracy can significantly influence our findings. Large bins can obscure important details making us to overlook key bounds that characterize the most discriminative patterns. However, small bins can lead to a noisy histogram with irrelevant bounds, which would unnecessarily complicate our analysis. The kind of histogram and the method of binning (e.g., equal width (default case), equal frequency (*percentogram*) or *bayesian blocks*) also change how we understand the data. We have to be careful about these factors to obtain a precise and meaningful histogram that can assist our analysis.

### 4.3.6. Finding the feature thresholds

As we have just seen, in order to compute the DNC vectors we first need the feature boundaries, $b_j = \{b_j^L, b_j^U\}$, for each feature $f_j$. This raises a new problem: how to find the feature bounds. In [49] manually set bounds were used, but this was later superseded in [50] where the bounds were adaptively computed through averaging over a sliding window. In our case, we propose to compute the feature bounds automatically for each patient and channel.

To tackle this problem we propose the use of an optimization tool that searches the space of possible bounds for all features with the goal of optimizing the *DR*. The chosen optimization tool is provided by the C++ library *dlib* [75], that among a wide range of machine learning algorithms, it also provides a global optimization function called "*global_function_search*". This function implements Lipschitz optimization [76] and aims at finding the global optima using as few

function evaluations as possible. For more information about this optimization tool, please refer to [75].

The optimizer requires two main inputs: i) the objective function, $OF$, and ii) the set of inequality constraints, $C$, that define a search space. The objective function receives the boundaries for each feature and returns the corresponding $DR$, $DR{=}OF(b_1, ..., b_5)$. The inequality constraints define the valid range in which the optimizer search for the optimal bounds (that result in the largest $DR$), $C = \{b_j^{Ll}, b_j^{Lu}, b_j^{Ul}, b_j^{Uu}\}; 1 \leq j \leq 5$, such that $b_j^{Ll} \leq b_j^{L} \leq b_j^{Lu}$ and $b_j^{Ul} \leq b_j^{U} \leq b_j^{Uu}$. In Figure 4.5 we can see in the y-axis the bounds and the domain in which they are constrained (green bands in the figure).

The objective function computes a $DR$ by first obtaining the $DNC_S$ and $DNC_Q$ resulting from the input bounds, $b_1, ..., b_5$. These DNC vectors define which distances of the Distance Matrix (see Figure 4.3) should be considered and from these the discrimination vector, $DV$, is computed. Finally, the objective function returns the maximum $DR_i$ in $DV$ and its corresponding $d_{max,i}$. If this maximum $DR_i$ is smaller than an threshold $DR_{min}$ (in our experiments set to 1.2), the function tries to find a batch of two patterns with a $DR_c > DR_{min}$ and if this is not possible, it tries with a batch of three patterns. The optimizer iterates by modifying the input bounds in the direction that maximizes $DR$ and stops when the user-defined maximum number of iterations is reached, returning at each iteration the smallest possible batch, $B^c$, that satisfies that $DR_c > DR_{min}$. At the end of the optimization process, for each one of the iterations for which a batch with $DR_c > DR_{min}$ was found, we know the optimal bounds, $b_j^L$ and $b_j^U$, for each feature, $f_j$, the resulting maximum $DR_c$ that correspond to the most discriminative batch of patterns $B^c$ and the corresponding discrimination threshold $d_{max,c}$.

Since the optimizer needs to read the Distance Matrix, $DM$, as well as the features of the epochs and patterns in several iterations, we precompute these constant matrices just once. The result of applying the 5 feature functions, $f_j$, to all epochs of a channel is stored in a matrix of size $5 \times n_e$ that we call the signal feature matrix, $V_S$. Formally, $V_{S_{j,k}} = f_j(E_k)$. Similarly, we can compute the query feature matrix $V_Q$, $V_{Q_{j,k}} = f_j(P_k)$, for all the patterns of the query, that is a matrix of $5 \times n_p$.

This lead us to finding the set of constraints, $C = \{b_j^{Ll}, b_j^{Lu}, b_j^{Ul}, b_j^{Uu}\}; 1 \leq j \leq 5$. The idea is to find a reasonable range of feature values exhibited by the seizures. Note that the five features of each pattern can be seen as a "physical" signature that identifies valid values for seizure amplitude, energy and integrated power in three bands. The collection of such signatures is stored in $V_Q$, comprising

the expected/valid features of the seizures. Using this collection to compute the range of valid bounds will help in filtering out (via DNC) irrelevant or misleading epochs of the signal. With this motivation, we define the constraints as follows: $b_j^{Ll} = min(V_{Q_j})$, $b_j^{Lu} = b_j^{Ul} = median(V_{Q_j})$ and $b_j^U u = max(V_{Q_j})$; $1 \leq j \leq 5$, being $V_{Q_j}$ the $j$-th row of the query feature matrix, $V_Q$.

### 4.3.7. Cleaning the signals: get rid of noise and artifacts

Any application relying on EEG data analysis benefits from signal filtering and artifact rejection[128]. By filtering, unwanted noise can be removed while preserving the underlying neural activity (improving the signal-to-noise ratio). In our proposal, a bandpass filter based on the *filter_data* algorithm [40] is applied to the signal. We employ a low cut-off frequency of 1.5 Hz, a widely adopted value in EEG analysis [77]. For the high cut-off frequency, we set it at 36 Hz to avoid filtering out potentially significant beta activity and the influence of gamma activity [125]. The filter is applied to the entire signal, $S$, and query, $Q$, and the result is stored in new variables $\hat{S}$ and $\hat{Q}$.

Additionally, EEG recordings are frequently affected by transient and non-stationary artifacts such as eye blinking, muscle activities, body movements, shifts in electrode position, and montage errors. Artifact rejection methods must be adaptable to the variability and specific characteristics of these different artifacts. [39, 89].

Our research relies on a recent and widely used artifact rejection method known as the Riemannian Potato Field (RPF) algorithm [10]. The idea is to characterize each epoch by its covariance matrix. This is, if we define a multi-channel epoch, $ME$, as a matrix of $nc$ channels and $e$ samples, then the covariance matrix of $ME$ is defined as $Cov(ME) = \frac{1}{e-1} ME \cdot ME^T$. The distance (similarity) of two covariance matrices is defined by the affine-invariant Riemannian (AIR) distance [93]. That way we can compute the covariance matrix of clean epochs, compute the geometric mean as a reference and flag as noisy/artifacted the epochs that are far (dissimilar) from this clean reference. This reference minimizes the dispersion (sum of square distances) and it can be seen as the center of mass of all the covariance matrices of the seizure regions. A z-score threshold, $z_{th}$, defines the region of acceptability that is known as the Riemannian Potato (RP) due to the aspect of its geometry.

When the number of channels, $nc$, is large, the corresponding dimension of the Riemannian geometry, $m = nc \cdot (nc + 1)/2$, becomes a problem: artifacts that cause significant variation in one dimension may be hidden by noise in all

other dimensions and go undetected. To address this issue, the RPF algorithm splits the number of channels in different sets to build several low-dimensional RPs, each designed to detect specific artifacts. In our research, we consider three types of artifacts: one RP is built for artifacts generated by eye blinks using frontal electrodes (FP1-F7, FP1-F3, FP2-F4, FP2-F8) readouts filtered in low frequencies (1-20Hz); a different RP for high-frequency (25-45Hz) artifacts in the occipital area (P7-O1, P3-O1, P4-O2, P8-O2); and a third one for low-frequency (0.5-3Hz) artifacts in all channels.

The RPF algorithm has been integrated in our solution as a Python script that returns a Do-Not-Compute (DNC) vector that identifies epochs as either usable (with 0's) or contaminated by artifacts (with 1's). The reference covariance matrices are computed out of the patterns in the query $Q$ that we use as clean EEG. Then, the covariance matrices for the epochs in $S$ are computed. The AIR distance in each RP is used to flag in the DNC the epochs that are far from the reference. Using $Q$ as the reference of clean EEG has the positive side-effect of filtering non-seizure epochs and help in the seizure detection, much in the way epileptic seizures are detected in [97].

## 4.4. Algorithmic details

In the previous section we have presented the main concepts that are used in the *PaFESD* algorithm following an order that eases the understanding of our proposal. In this section, we will summarize the algorithm itself following the order in which these concepts are really implemented. We will start by explaining the input arguments that are used to feed the algorithm. Then, we provide a high-level view of the algorithm itself, including the validation process.

We have already mentioned several parameters that can be set to tune the algorithm's behavior. We summarize in Table 4.3 these parameters and their default values (the ones used in the experimental evaluation section). These values have been selected based on the existing literature we have reviewed and our own experience after exploring different options. The multichannel time series of the patient $p$, $T^p$, is read from the CHB-MIT database, along with the metadata, $md$, that identify the seizure regions. With this input data and the arguments of Table 4.3, the method that we propose is summarized in Algorithm 3.

As we can see, the algorithm is divided in 4 stages: i) clean() takes care of filtering the time series $T$ and of flagging in $DNC_S$ the epochs identified as artifacts; then for each channel of $T$: ii) we first split the channel in the

| Sym. | Default | Description |
|---|---|---|
| $T^p$ | – | EEG time series of patient p |
| $e$ | 1280 | Num. of samples of the epoch/pattern sliding windows |
| $s$ | 256 | Num. of samples of the epoch/pattern stride |
| $w$ | 16 | Size of DTW warping window |
| $d_{th}$ | 1.05 | Tolerance factor applied to $d_{max}$ |
| $n_o$ | 650 | Number of iterations of the optimizer |
| $DR_{min}$ | 1.2 | Min acceptable $DR$ |
| $n_b$ | 3 | Max size of the batch $B^c$ |
| $z_{th}$ | 5 | RP threshold to discriminate artifacts |

Table 4.3: Main parameters of the *PaFESD* algorithm

**Algorithm 3:** *PaFESD* pseudocode

**Input:** $T$, $md$ // Time Series and metadata from EEG dataset
**Output:** *quality_metrics* // Quality metrics of the algorithm
/* Filter and artifact rejection */
$\hat{T}, DNC_S = clean(T, md)$
/* For each channel in the time series */
**for** $\hat{S}^c$ ***in*** $\hat{T}$ **do**
  /* Compute required variables */
  $\hat{S}^{c_t}, \hat{S}^{c_v}, \hat{Q}, DNC_Q, DM, V_S, V_Q = preproc(\hat{S}^c, DNC_S, md)$
  /* Get the batch of discriminative patterns */
  $B^c = find\_patterns(DNC_S, DNC_Q, DM, V_S, V_Q)$
  /* Report precision, recall and $F_1$ score */
  $quality\_metrics = validation(\hat{S}^{c_v}, DNC_S, md, B^c)$
**end**
**return** *quality_metrics*

training, $\hat{S}^{c_t}$, and validation, $\hat{S}^{c_v}$, sections, build the query, $\hat{Q}$, by concatenating the seizures used for training, update auxiliary vectors ($DNC_S$ and $DNC_Q$) and compute auxiliary matrices ($DM$, $V_S$ and $V_Q$); iii) then we find the batch, $B^c$, with up to $n_b$ patterns; and iv) finally these patterns are assessed using the validation section of the channel. We delve into each one of these stages next.

### 4.4.1. Cleaning the time series

This stage carries out the data loading, data filtering and artifact removal as can be seen in the pseudocode of Algorithm 4.

**Algorithm 4:** Cleaning phase: $\hat{T}, DNC_S = clean(T, md)$

```
   Input: T, md                    // EEG time series from a patient and metadata
   Output: T̂, DNC_S                // Filtered time series and DNC vector
   /* Apply bandpass filter to all channels in T                              */
1  T̂ = filter(T, lowcut = 1.5Hz, highcut = 36Hz)
   /* Initialize DNC vector with zeros                                        */
2  DNC_S = zeros()
   /* For each RP configuration                                               */
3  for RP_conf in {eye-blink, occipital, general} do
       /* Covariance matrices of the seizure and non-seizure regions          */
4      {Cov(ME+), Cov(ME−)} = covariances(RP_conf, T, md)
       /* Clean covariance reference                                          */
5      Ref = center_of_mass({Cov(ME+)})
       /* Update DNC_S                                                        */
6      DNC_S = DNC_S ∧ (AIR_distance(Cov(ME), Ref) > z_th)
7  end
8  return T̂, DNC_S
```

In line 1 we apply a bandpass filter to the time series $T$ (all channels) to remove frequencies outside the range $[1.5, 36]$ Hz. Then, for each one of the three Riemannian Potato configurations, we compute the covariance matrices of the seizure, $\{Cov(ME^+)\}$, and non-seizure, $\{Cov(ME^-)\}$, regions. These seizure and non-seizure regions are identified using the metadata *md*. The covariance matrices are computed as we explained in Sec. 4.3.7. Using the covariance matrices of the seizure regions we can compute a clean reference, *Ref*, as the covariance matrix that represents the centroid of the covariance cloud.

Finally, in line 6, a vector operation updates the (initially zeroed) $DNC_S$ vector, flagging the epochs that are identified as artifacts. These are the epochs with a covariance matrix that are at a distance from the reference covariance matrix, *Ref*, (using affine-invariant Riemannian -AIR- distance) greater than $z_{th}$. The *clean()* function returns the filtered time series, $\hat{T}$, and the DNC vector, $DNC_S$, that identifies all the artifacted epochs.

### 4.4.2. Channel preprocessing and pattern finding

The main algorithm (see Algorithm 3) continues by processing each channel, $\hat{S}^c$, of the time series, $\hat{T}$. In line 3, each channel gets preprocessed in order to compute auxiliary variables that are needed by subsequent functions. First, each channel is split into two sections: the training section, $\hat{S}^{c_t}$, and the validation section, $\hat{S}^{c_v}$. The training section is used to find the batch of patterns, $B^c$, that

will be used in the validation section to detect seizures. The query, $\hat{Q}$, is built by concatenating the seizures of the training section. The vector $DNC_Q$, the distance matrix, $DM$, and the feature matrices, $V_S$ and $V_Q$, are computed in this stage as well, as explained in Sec. 4.3.3 and Sec. 4.3.6, respectively.

Using these auxiliary variables we can now find the batch of discriminative patterns as we sketch in Algorithm 5. The objective function, $OF$, is a function pointer or functor parametrized by the feature bounds (variables $b_1, \cdots b_5$). The objective function also requires initial DNC vectors and the constant matrices ($DM$, $V_S$ and $V_Q$) as we see in line 1. Note that this function does not need the signal nor query samples because the required information has been previously gathered and stored in the input vectors and matrices. The constraints, $C$, are computed in line 2 and passed to the optimization function along with $OF$, as described in Sec. 4.3.6.

**Algorithm 5:** Find patterns phase:
$B^c = find\_patterns(DNC_S, DNC_Q, DM, V_S, V_Q, md)$

**Input:** $DNC_S$, $DNC_Q$, $DM$, $V_S$, $V_Q$ // Auxiliary variables and matrices
**Output:** $B^c$ // Batch of discriminative patterns
/* Bounds, DNCs, DM and feature matrices */
1 $OF \leftarrow DR{=}Object\_Func(b_1, \cdots b_5, DNC_S, DNC_Q, DM, V_S, V_Q)$
/* Constraints for the optimizer */
2 $C \leftarrow Constraints(V_Q)$
/* Call optimizer with $OF$, $C$ and number of iterations */
3 $Candidates = Global\_Function\_Search(OF, C, n_o, n_b)$
/* Select one batch from the candidates */
4 **return** $B^c = Batch\_selection(Candidates)$

Another category is added to the set based on the $lb$ parameter described in the section 4.4. This new parameter is incorporated in the iteration space because of the good results of these types of filtering techniques in other studies [50, 139].

As we discussed in Sec. 4.3.4 and will assess in the experimental section, using batches with more than one pattern can improve the precision and reduce the number of false positives of the seizure detection algorithm. Thus, the function "Global_Function_Search" also receives as input parameters the number of iterations, $n_o$, and the maximum size of the batch, $n_b$. It returns up to $n_o$ batches, $B^c$, fulfilling that $DR_c > DR_{min}$ or it is the batch with the largest $DR_c$ for the given bounds and $n_b$. The batches are stored in a set, *Candidates*, that is returned by the function. The information included in each batch is the following: the list of pattern IDs, the $DR$ of the batch, $DR_c$, the discrimination threshold, $d_{max,c}$, and the bounds, $b_1, \cdots, b_5$, used in that iteration of the optimizer.

In the experimental section we validate all batches in the *Candidates* set and call this strategy "Oracle" because it assesses all the batches found. However, in practice, one of these candidates should be identified and validated. To that end we call a *Batch_selection()* function that selects one batch using the bounds, the *DR* and the discrimination threshold of the *Candidates*. These bounds position each batch in a 10D space (five bounds, $b_j$, with two limits each, $\{b_j^L, b_j^U\}$). The selection policy (named "Centroid") first computes the weighted by *DR* center of mass (centroid) of the candidates and then selects the nearest neighbor candidate.

### 4.4.3. Validation

Once we have the batch, $B^c$, we can assess it using the validation section of the channel, $\hat{S}^{c_v}$. The validation phase is summarized in Algorithm 6. As we can see, in addition to $B^c$ and $\hat{S}^{c_v}$, we also have to read the metadata, $md$, that identifies the seizure and non-seizure regions of the channel and the $DNC_S$ that comes from the cleaning stage where we did the signal filtering and artifact rejection of the whole channel $S^c$ (including $S^{c_t}$ and $S^{c_v}$). First, in line 1 we clear the counters, $TP, FP, TN, FN$, that keep the number of true positives, false positives, true negatives and false negatives, respectively. Then, we extract the required information from the batch, $B^c$, in line 2. This information includes the feature bounds, $Bounds = b_1, ..., b_5$, the $n_b$ patterns of the batch, $\{P_j\}$, and the discrimination threshold, $d_{max,c}$. The bounds are used to update the $DNC_S$ vector in line 3 using the function *filter_S* that computes the five features of each epoch in $\hat{S}^{c_v}$ and flags those out of the bounds.

Then, for each valid epoch (i.e., $DNC_S[i] == 0$), $E_i$, in the validation section, $\hat{S}^{c_v}$, we compute the DTW distance between the epoch and all the $n_b$ patterns in the batch. If the minimum DTW distance is smaller than $d_{max,c} \cdot d_{th}$, we count a seizure/positive or a non-seizure/negative otherwise. Using the metadata, $md$, we can update the $TP, FP, TN$, and $FN$ counters. Finally, we return the quality metrics, *quality_metrics*, out of these counters.

## 4.5. Experimental Evaluation

The following experiments have been conducted on a Cluster Lenovo ThinSystem SR645 that features 160 nodes, each with two AMD EPYC 7H12 processors (128 cores/node). Each processor runs at 2.60GHz (base, up to 3.3GHz) and each node has 512GB of RAM. The code has been implemented in Python 3, but some time-consuming functions (computation of the DM, batch searching and

**Algorithm 6:** Validation phase: $quality\_metrics = validation(\hat{S}^{c_v}, DNC_S, md, B^c)$

**Input:** $\hat{S}^{c_v}$, $DNC_S$, $md$, $B^c$ // Validation signal, DNC vector, metadata and batch
**Output:** $quality\_metrics$ // Quality metrics of the algorithm

/* Counters initialized to zero */
$TP = 0; FP = 0; TN = 0; FN = 0$
/* Extract required information from $B^c$ */
$Bounds, \{P_j\}, d_{max,c} = get\_batch\_content(B^c)$
/* Update $DNC_S$ using the features and bounds */
$DNC_S = DNC_S \wedge filter\_S(\hat{S}^{c_v}, Bounds)$
/* For each epoch in the validation signal */
**for** ***each*** $E_i \in \hat{S}^{c_v}$ ***where*** $DNC_s[i] == 0$ **do**
  /* Check the min DTW distance from $E_i$ to all patterns in the batch */
  **if** $min_{j=1}^{n_b}(d(P_j, E_i)) < d_{max,c} \cdot d_{th}$ **then**
    $TP, FP = check\_seizure(TP, FP, i, md)$
  **else**
    $TN, FN = check\_no\_seizure(TN, FN, i, md)$
  **end**
**end**
/* Return the Precision, Recall and $F_1$ score */
**return** $quality\_metrics = PRE\_SEN\_F1(TP, FP, FN)$

optimizer) are actually C++ procedures that have been wrapped with Pybind11 or Cython so that they can be called from Python. Coarse-grained parallelism is exploited (among patients and channels), as well as medium-grained (computation of the DM) and fine-grained parallelism (by SIMDimizing the computation of batches of DTW distances). Table 4.2 (see Sec. 4.3.1.1) summarizes the main characteristics of the CHB-MIT dataset. Although most patients have 23 channels, patients 18 and 19 only have 18 channels, so we evaluate these 18 common ones: C3-P3, C4-P4, CZ-PZ, F3-C3, F4-C4, F7-T7, F8-T8, FP1-F3, FP1-F7, FP2-F4, FP2-F8, FZ-CZ, P3-O1, P4-O2, P7-O1, P8-O2, T7-P7, and T8-P8. Considering the 24 patients, we evaluate 432 channels with each of the following strategies: Baseline (only DTW), Baseline+Cleaning, Features (only signal features) and our proposal (*PaFESD*). Table 4.4 summarizes the mean values that each approach achieves for our quality metrics (PRE, SEN and $F_1$) considering up to 3 patterns per batch. The mean times per patient for training (cleaning, preprocessing and pattern finding stages) and validation (validation phase) are also shown (in seconds). These results are analyzed in more details in the next subsections.

### 4.5.1. Baseline

To serve as a baseline we have an implementation based on only DTW distances (without filtering, artifact rejection, or the use of signal features). This is a similar approach to the one proposed by [117] but with a major difference: in [117] the batch is manually selected by visually inspecting a channel for each patient. The discrimination threshold, $d_{max}$, is also artificially chosen after computing all DTW distances for the whole $S^c$, so that no $FP$ are reported. This is equivalent to train on the whole channel $S^c$ ($S^{c_t} = S^c$). In contrast, in our baseline algorithm, for each channel $c$, we only train with $S^{c_t}$ and then validate the automatically obtained batch, $B^c$, using the remaining $S^{c_v}$.

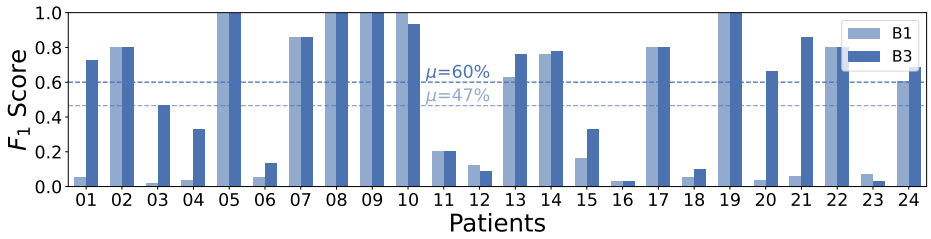



Figure 4.6: Baseline F1 score

The plot in Figure 4.6 shows the $F_1$ metric for the described *Baseline* version and the most discriminative channel of each of the 24 patients in CHB-MIT using two different batch sizes: B1 (1 pattern per batch or no batching) and B3 (up to 3 patterns per batch). As we can see, the *Baseline* algorithm achieves an average $F_1$ score of 47% on the whole dataset for B1. This low value indicates that the baseline algorithm is not able to detect seizures accurately. The results also show that the *Baseline* algorithm performs better with a batch size of up to

| **Metric** | *Baseline* | *Baseline* +Cleaning | *Features* | *PaFESD* Centroid | *PaFESD* Oracle |
|---|---|---|---|---|---|
| PRE | 66.9% | 79.9% | 53.2% | 98.3% | 99.5% |
| SEN | 82.0% | 94.0% | 95.9% | 99.6% | 100.0% |
| $F_1$ | 60.0% | 81.0% | 59.2% | 98.9% | 99.7% |
| train. t (sec.) | 338.8 | 404.4 | 595.3 | 726.8 | 726.8 |
| val. t (sec.) | 0.55 | 0.50 | 82.4 | 0.06 | 102.8 |

Table 4.4: Comparison of quality metrics, and mean times of training, and validation for different approaches

3, achieving an average $F_1$ score of 60%. From Table 4.4 we see that the precision metric -PRE- is the one with low performance (just 66.9%): it is, this approach tends to return a relevant number of *FP* or false alarms.

It is noteworthy that for patients 10, 12, and 23 in Figure 4.6, $F_1$ slightly decreases when moving from no-batching to a batch of up to three patterns. Although this may seem counterintuitive, larger batches that were selected as discriminative during training may not necessarily be optimal for validation. This is because the larger the batch, the more likely it is to find a smaller non-seizure nearest neighbor, $nn^-$ in this phase. In some cases this can lead to more *FP* if this $nn^-$ is smaller than $d_{max}$.

### 4.5.2. Baseline + Cleaning

To assess the impact of the cleaning strategies (filtering and artifact rejection, see Sec. 4.3.7), we assess now a *Baseline* plus *Cleaning* implementation, whose $F_1$ results are shown in Figure 4.7 for the most discriminative channel of each of the 24 patients, and B1 and B3 batch sizes.

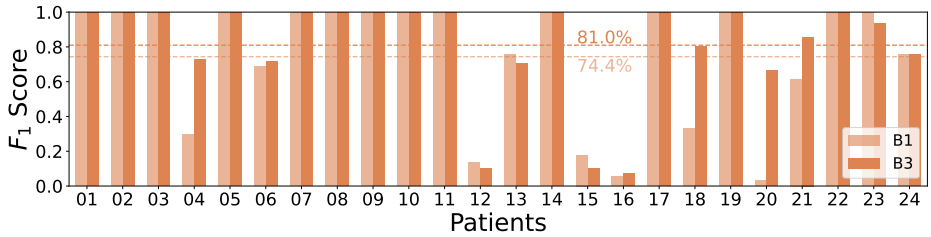



Figure 4.7: Baseline+Cleaning $F_1$ score

As we can see, the enhanced algorithm yields significantly better results than the baseline, with an average $F_1$ score of 74% and 81% for the B1 and B3 batch sizes, respectively. Notably, the B3 batch size achieves $F_1$ scores exceeding 90% for several patients, demonstrating the utility of EEG signal cleaning procedures for epileptic seizure detection. To gain further insight into the impact of the cleaning stage, we compare the results of *Baseline* and *Baseline+Cleaning* with B3 batch size in Table 4.4. Both the precision and sensitivity metrics improve when incorporating the cleaning stage: in particular it achieves 100% of precision (no *FP*) and 100% of sensitivity (all seizures are correctly detected) for 13 out of 24 patients.

In summary, the integration of cleaning techniques into the baseline algorithm

substantially boosted its seizure detection performance, adding an average of 21% to the $F_1$ score. This is mainly due to the cleaning rate (percentage of 1's in the DNC vector) that is approximately 76% on average, representing a significant amount of non-seizure epochs discarded from the analysis thanks to the cleaning stage.

### 4.5.3. Features

Now, we evaluate the behavior of a different version of our algorithm that only leverages signal features (see Sec. 4.3.5) in order to identify epileptic seizures. We compare the results of this algorithm, referred to as *Features*, with *Baseline* and *Baseline+Cleaning* algorithms. Note that this "only-features" algorithm does not include the cleaning stage nor the pattern selection based on DTW distances (as the baseline does). Instead, it focuses solely on identifying each feature range of values that characterize the epileptic seizures. With this experiment we seek to see to what extent the epoch features provide enough information to identify seizures.

The goal in this implementation is to automatically obtain a DNC vector that filters out all non-seizure epochs, $E^-$, but keeps at least one seizure epoch per seizure subsequence, $E^{+i}$. To that end, the feature bounds, $b_1, ..., b_5$, have to be automatically identified so that such DNC can be obtained in the training section of the channel, $S^{c_t}$. This bound identification is conducted inside the optimizer by using a different objective function, $OF$, that instead of returning a $DR$ (currently not possible as DTW distances are not computed), it returns values that identify three different situations. This value is 0 if there exists at least one seizure, $i$, without at least one epoch in $E^{+i}$ surviving the feature filtering. When the value is 0, we say that the corresponding DNC is 'invalid' because it fails to detect at least one seizure. The $OF$ returns 1 if the used bounds result in a valid DNC vector (all seizures are detected) but also epochs in $E^-$ have survived (i.e., there are $FP$). Finally, the value 2 is returned when obtaining a valid DNC vector and all non-seizure epochs has been discarded (no $FP$). The optimizer iterates now $n_o = 4000$ times and returns the bounds when the $OF$ returns 1 and 2.

This approach yields up to $n_o$ distinct bounds. As discussed in Sec. 4.4.2, our policy, named "Centroid" selects and validates a single candidate solution by computing the centroid of all reported bounds and taking the nearest neighbor. During validation, the obtained bounds of the solution are used to compute the DNC vector for the validation set, $S^{c_v}$.

Figure 4.8 shows the $F_1$ score of this *Features* implementation along with the

previously discussed *Baseline* and *Baseline+Cleaning* alternatives. We can see that *Features* results in an average $F_1$ score of 59.2%. In fact, only 29% of the patients achieve an $F_1$ score above 90%. As we can see in Table 4.4, *Features* shows the worst precision when compared to *Baseline* and *Baseline+Cleaning*, indicating that a high number of false positives are reported for most patients. However, it also shows an improvement of the sensitivity when compared to the other approaches, which seems to point to a better accuracy in the detection of seizures.

In our evaluation of the *Features* approach many valid candidates were found during training. Actually, out of the 4000 iterations conducted by the optimizer, on average, 2051 (51%) were categorized as "valid" solutions (with a valid DNC vector). The "Centroid" policy selected just one of these valid solutions. It should be mentioned, that among the valid solutions, 425 (11% of the total) yielded a DNC vector that discarded all non-seizure epochs, demonstrating the potential for effective seizure identification, in case it could be guaranteed that an appropriate batch would be chosen.

It is worth noting that, since there is no cleaning stage in this implementation, there is not a DNC cleaning rate (percentage of 1's in the DNC). However, the algorithm filtered around 94% of non-seizure epochs on average, which represents a significant amount of signal that can be discarded because it does not contain seizures. However, this approach lacks precision to detect the pattern (shapes) of the seizures because it is not able to clearly distinguish between a *TP* (real seizure) and a *FP* (false alarm) from a surviving non-seizure epoch.

### 4.5.4. PaFESD: Baseline+Cleaning+Features

Finally, we assess the effectiveness of our proposal, *PaFESD* when all three stages (cleaning, pattern matching based on DTW distance and features) are combined to exploit the synergies between them. In Figure 4.8 we compare the *PaFESD* $F_1$ results with the previous approaches for the most discriminative channel for each patient, B3 batch sizes (although in the end, $n_b$ is always 1 or sometimes 2), and the "Centroid" policy for selecting a batch from the $Candidates$ set.

We see that our proposal, which effectively combines the different strategies, *Baseline+Cleaning+Features*, improves the average $F_1$ score up to 98.9%. In *PaFESD*, the cleaning rate due to the cleaning stage is 76%, that combined with the features filtering rises the percentage of non-seizures epochs effectively filtered over 97%. We can see that this percentage of filtered non-seizure epochs

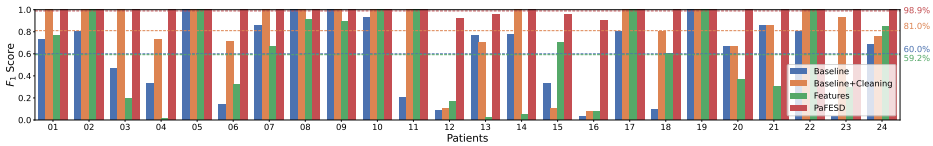



Figure 4.8: *Baseline-B3*, *Baseline+Cleaning-B3*, *Features*, and *PaFESD-B3* $F_1$ score

has increased slightly with respect to the one obtained with the features-only approach (94%). But the relevant synergy happens when these stages are incorporated in a pattern matching strategy based on DTW distances as we propose: this will significantly improve the accuracy to select the pattern of the seizures to correctly detect them without false positives, as we will see next.

To provide more details, in Table 4.5 we show for each patient: the discriminative channel, the number of seizures, the percentage of the total EEG signal used during the training, the number of seizures required for training, the batch size, and the results obtained by the "Centroid" policy. For this policy, we list the total number of seizures detected ($TP$), the false alarms ($FP$) and False Positives per Hour (FpH). We conservatively compute FpH by assuming that 0 $FP$ in $x$ hours of EEG data result in FpH=$1/x$ (i.e. in the worst case a $FP$ can arise just after the $x$ hours). The average FpH is 0.043, but considering that we only have 11 $FP$ in 980h of total EEG data, a more realistic value is FpH=0.011. Finally, we show the $F_1$ score. Patients with $F_1$ score below 100% are highlighted in red.

We can see in Table 4.5 that we get a perfect $F_1$ (no $FP$ nor missed $TP$) for 20 patients (83% of the dataset). However, for patients 12, 13, 15 and 16 the number of $FP$ can be set at one, two, or even 7 for patient 12. Or even we can miss a seizure for patient 16. In order to see if we have room for improving our results for these four challenging patients, we perform and additional study by setting up the "Oracle" policy, which evaluates all batches returned by the optimizer and selects the best one after validation. Table 4.4 shows a comparison of the quality metrics for both "Centroid" and "Oracle" policies, finding that "Oracle" improves the precision to 99.5%, the sensitivity to 100% and $F_1$ score to 99.7%. This means that, under this training conditions and when selecting the ideal batch, we could reduce the number of $FP$ for all patients, but most importantly we would not miss any true seizure (not even for patient 16). These four challenging patients share that $10 \leq N_z \leq 40$, thus having trained with only $N_z^t \leq 4$ (between 20% and 67% of $N_z$) has also an impact in the detection quality metrics. We perform an additional evaluation increasing $N_z^t$ to 7, 5 an 6,

| p | Ch. | $N_z$ | Train % | $N_z^t$ | $n_b$ | TP | FP | FpH | $F_1$ |
|---|---|---|---|---|---|---|---|---|---|
| 01 | F3-C3 | 7 | 20% | 2(29%) | 1 | 7 | 0 | 0.025 | 100% |
| 02 | C3-P3 | 3 | 20% | 2(67%) | 1 | 3 | 0 | 0.029 | 100% |
| 03 | F3-C3 | 7 | 20% | 2(29%) | 1 | 7 | 0 | 0.026 | 100% |
| 04 | F4-C4 | 4 | 20% | 2(50%) | 1 | 4 | 0 | 0.006 | 100% |
| 05 | C4-P4 | 5 | 20% | 2(40%) | 1 | 5 | 0 | 0.026 | 100% |
| 06 | C3-P3 | 10 | 20% | 2(20%) | 1 | 10 | 0 | 0.015 | 100% |
| 07 | C4-P4 | 3 | 20% | 2(67%) | 1 | 3 | 0 | 0.015 | 100% |
| 08 | C3-P3 | 5 | 20% | 2(40%) | 1 | 5 | 0 | 0.050 | 100% |
| 09 | C3-P3 | 4 | 20% | 2(50%) | 1 | 4 | 0 | 0.015 | 100% |
| 10 | C3-P3 | 7 | 20% | 2(29%) | 1 | 7 | 0 | 0.020 | 100% |
| 11 | C3-P3 | 3 | 20% | 2(67%) | 1 | 3 | 0 | 0.029 | 100% |
| 12 | C4-P4 | 40 | 20% | 4(10%) | 2 | 40 | 7 | 0.304 | 92% |
| 13 | FP1-F7 | 12 | 30% | 4(33%) | 2 | 12 | 1 | 0.030 | 96% |
| 14 | C3-P3 | 8 | 20% | 2(25%) | 1 | 8 | 0 | 0.038 | 100% |
| 15 | T7-P7 | 20 | 20% | 4(20%) | 1 | 20 | 2 | 0.050 | 95% |
| 16 | C4-P4 | 10 | 50% | 4(40%) | 2 | 9 | 1 | 0.053 | 90% |
| 17 | C3-P3 | 3 | 20% | 2(67%) | 1 | 3 | 0 | 0.050 | 100% |
| 18 | P8-O2 | 6 | 30% | 2(33%) | 1 | 6 | 0 | 0.029 | 100% |
| 19 | C4-P4 | 3 | 20% | 2(67%) | 1 | 3 | 0 | 0.034 | 100% |
| 20 | C3-P3 | 8 | 30% | 2(25%) | 1 | 8 | 0 | 0.037 | 100% |
| 21 | C3-P3 | 4 | 20% | 2(50%) | 1 | 4 | 0 | 0.031 | 100% |
| 22 | C4-P4 | 3 | 20% | 2(67%) | 1 | 3 | 0 | 0.032 | 100% |
| 23 | F3-C3 | 7 | 25% | 2(29%) | 1 | 7 | 0 | 0.038 | 100% |
| 24 | F3-C3 | 20 | 20% | 4(20%) | 2 | 20 | 0 | 0.048 | 100% |
| **Mean** | - | 8 | 22% | 2(29%) | 1 | 8 | 0 | 0.043 | 98.9% |

Table 4.5: Results for *PaFESD* algorithm

for patients 13, 15 and 16, respectively, and for the "Centroid" policy we now get a perfect $F_1$ for these patients, increasing the average $F_1$ from 98.9% to 99.6%.

## 4.5.5. Discussion

### 4.5.5.1. Sensitivity analysis

Our aim at fixing $N_z^t \leq 4$ or that $n_t$ represents up to 50% of the signal used for training, has been the reduction of the model's complexity and its training and validation times. Our analysis reveals that the algorithm's performance is robust across different patients. This is a strong indicator that our model is not underfitting. Underfitting occurs when a model is too simple to capture all the relevant patterns in the data, resulting in poor sensitivity, which is not the case. However, our findings also suggest that for some challenging patients, *PaFESD*'s $F_1$ score may slightly degrade if trained with not enough seizures. These cases raise concerns about the potential overfitting to the training data in our approach. For instance, for patient 12, the algorithm was able to accurately detect all 40 seizures but with seven *FPs*. Increasing $N_z^t$ to 6 for this patient just reduces in 1 the number of *FP*. This could be indicative of overfitting, where the model is so finely tuned to the training data that it starts to identify noise (false positives) as seizures.

To test whether we have overfitting, we can incrementally increase the number of iterations of the optimizer, $n_o$, and measure the resulting quality metrics. If the metrics do not improve, this indicates that the model is overfitting to the training data. In this case, we can increase the size of the signal used for training, $n_t$, while maintaining the number of seizures, $N_z^t$, or also increase $N_z^t$. Increasing $n_t$ can help the model better generalize to new, unseen data. Increasing $N_z^t$ allows more data to be gathered from a wider range of seizures, which can help the model generalize better to different types of seizures. However, both approaches also increase the training and validation times. In any case, we would like to explore this issue more thoroughly as future work.

| **Metric** | $es = 2$ | $es = 3$ | $es = 4$ | $es = 5$ | $es = 6$ |
|---|---|---|---|---|---|
| $F_1$ | 99% | 98.6% | 98.8% | 98.9% | 98.8% |
| train. t (sec.) | 714.4 | 777.1 | 905.3 | 726.8 | 579.5 |
| val. t (sec.) | 0.106 | 0.078 | 0.071 | 0.060 | 0.101 |

Table 4.6: Impact of different epoch sizes (in seconds, $es$) on quality metric $F_1$, and on mean times of training and validation.

Other issue that we have addressed in our experiments is the effect of different epoch sizes (from 2 to 6 seconds) on our seizure's detection approach, *PaFESD*, by studying its impact on the quality metric $F_1$ score, but also on the training and validation times. The selection of 2 and 6 seconds as the epoch size extremes is justified based on common practice in EEG analysis, where shorter epochs can capture rapid changes in the signal, while longer epochs may provide more context for detecting patterns.

The new results can be seen in Table 4.6. They show that the $F_1$ score remains consistently high across all epoch sizes, indicating that *PaFESD* exhibits robust seizure detection performance. Specifically, the highest mean $F_1$ score was achieved with a 2-second epoch size (99%), while the lowest was with a 3-second epoch size (98.6%), showing minimal variation overall.

However, from Table 4.6 we observe differences in the computational times. For instance, longer epochs such as 5 seconds result in shorter validation times. This is due to fewer epochs being checked, although increasing the size of the epoch increases the processing time per epoch (more detail in the next subsection), which can counteract the reduction due to fewer epochs, as we see for the the 6-second size. Therefore, there is a trade-off size for which the validation time is minimum, while the $F_1$ score is still high: the epoch size of 5 seconds, that is the default value in our study.

Training times also vary, with 2-second epochs requiring 714.4 seconds and 6-second epochs requiring only 579.5 seconds. There is again a trade-off between epoch size and computational cost. The size of the epoch has more impact in the training time than the number of epochs (till 4-second epochs), and the other way around beyond this point.

We have also conducted a limited randomization analysis focused on the selection of the training subsequence. Our analysis now involves training using 1000 different random subsequences for each patient. The results show a mean $F_1$ score of 99.2% with a standard deviation of 2%, which are slightly better than the original results obtained using the first available subsequence, indicating high consistency across different training samples. However, we observed some variations in performance for a small subset of patients, with trade-offs between *TP* and*FP*. For example, in the case of chb12, the number of *TP* decreased from 40 to 39, while the number of *FP* decreased from 7 to 1. Similarly, for chb15, the number of *TP* decreased from 20 to 19, while the number of *FP* decreased from 2 to 1. This analysis also revealed a correlation between seizure duration and detection probability. Seizures lasting less than 50 seconds have the highest probability of non-detection, with about 60% of undetected seizures being shorter

than 50 seconds, and approximately 80% under 75 seconds. Conversely, seizures lasting more than 150 seconds have a very low probability of going undetected. These findings suggest that while our algorithm demonstrates robustness across various training subsets, there is room for improvement in detecting shorter-duration seizures. This limited statistical analysis provides initial evidence of our algorithm's robustness and highlights areas for potential enhancement. A more comprehensive randomization study including exploration of various tuning parameters (the size of the training section, number of seizures to train, starting point of the training section) and strategies to improve short-duration seizure detection (such as adaptive windowing techniques or signal fusion approaches) remains an important direction for future work.

#### 4.5.5.2. Complexity analysis

Table 4.4 gives also some insights about the complexity of the evaluated approaches when considering the default 5-second epoch size. *Baseline* is the one with the smallest training times. By incorporating the cleaning stage in *Baseline+Cleaning*, the training times increase by 19%, but the filtering of epochs improves slightly the validation times. *Features* is a time consuming approach, both in training and validation stages. Our proposal, *PaFESD*, is the most costly approach for the training, but the fastest in the validation, thanks to the more effective filtering of non-seizure epochs and the efficient DTW distance computation. In fact, the latency to detect a seizure (detection delay) in our approach is 15 seconds, which could be a reasonable starting point when porting our algorithm to a wearable, although this is another issue that we will explore as a future work.

While the training can be carried out offline, we need real-time performance in the seizure detection version of *PaFESD*, which consists of the following steps: i) sample the EEG channel; ii) compute five features, $f_j$, to discard out of bound epochs; and iii) compute up to three DTWs to compare the surviving epochs with up to three discriminative patterns of the batch.

In step ii), the computation of features $f_1$ (peak-to-peak) and $f_2$ (energy) exhibit linear complexity $O(e)$, so times are negligible ($e = 1280$ samples for 5-second epochs). Band power features, $f_3$, $f_4$ and $f_5$, require computing the Welch's method -with complexity $O(e{\cdot}log(win\_size))$- to calculate the $e/win\_size$ DFTs for each Welch's window. We use Welch's window size of 512 samples, which results in complexity $O(e \cdot log(512))$ (also linear). Finally, in step iii), each DTW has complexity $O(e \cdot 2 \cdot w)$ ($w$ is the warping window size, 16 in our experiments).

As mentioned previously, our final goal is porting our seizure detection to an ultra-low-power wearable device. Preliminary results obtained on a Raspberry Pi 4 demonstrate that our algorithm can process/check more than 60 epochs per second which is 60x more than required to get real-time since, with stride $s = 256$ samples, we receive a new epoch every second. We are now implementing the seizure detection on a RISC-V ESP32-C3 low-power microcontroler that is slower but also consumes way less energy.

#### 4.5.5.3. Further improvements

Currently, in our approach, the mean latency to detect a seizure (detection delay) for the discriminative channel is 15 seconds. A potential enhancement could be to adapt our algorithm to prioritize finding discriminative patterns close to the seizure onset zone and select channels accordingly, which would reduce the detection delay. Moreover, as future work we are studying to tailor our method to identify discriminative patterns in the pre-ictal signal in order to detect seizures as earliest as possible.

## 4.6. Conclusions and Future Work

Our research presents a new method for the detection of epileptic seizures in EEG signals. We divert from previous works in that we tightly couple discrete-time signal analysis and time series pattern matching based on DTW distance. The method has been tested on the CHB-MIT EEG database and has achieved a precision of 98.3% and a sensitivity of 99.6%, outperforming other related works.

As future work, in the near term we plan to further optimize our implementation porting all time-consuming functions to C++ and using parallelization techniques to speed up the execution of the training and validation phases. With a faster implementation we can conduct a more exhaustive sensitivity analysis for the different parameters of the algorithm and assess their influence on the quality metrics, as well as a comprehensive statistical testing to demonstrate the robustness of our approach. Enhancements such as adapting our method to identify discriminative patterns in the pre-ictal signal, close to the seizure onset zone, will be explored as well. We are also working in the implementation of the seizure detection on a low-power RISC-V microcontroller to evaluate its real-time performance and energy efficiency on a case of use based on a wearable.

# 5 Conclusions

In this thesis, we explored advanced methodologies to enhance massive sensor-based data processing, focusing on algorithmic improvements, optimized data structures and optimizations that exploit device-specific architectural features using as cases of study sensor-based applications such as determine the Digital Terrain Model (DTM) in LiDAR point clouds, and epileptic seizure detection using EEG.

First, in Chapter 2, we discussed the significant shift towards digitalization, emphasizing the role of sensor-based data analytics in achieving operational excellence and competitive advantage. We highlighted the challenges posed by the data deluge, including the need for advanced analytical capabilities to manage the scale, variety, and velocity of data. This background set the stage for understanding the critical importance of data-driven decision-making in modern sensor-based applications.

Next, in Chapter 3, we focused on accelerating the OWM algorithm (Overlap Window Method), a crucial component in Digital Terrain Model (DTM) applications. Our research addressed the prohibitive processing times often associated with large LiDAR point clouds. We transitioned the algorithm from its original R implementation, which imposed serious limitations on processable data sizes, to a highly optimized C++ version. This transition involved exploring various optimization strategies tailored to the specific capabilities of CPU, iGPU, and dGPU devices. We leveraged state-of-the-art heterogeneous programming environments, including OpenMP, TBB, SYCL, and CUDA, to develop parallel implementations for commodity CPU and GPU devices. Our optimizations focused on minimizing memory accesses and refining data structures, resulting in remarkable performance improvements. The CPU and GPU versions achieved speedups





of up to 19x and 83x, respectively, compared to an OpenMP baseline utilizing eight CPU cores. Notably, these optimization techniques, encompassing careful data structure selection, effective parallelization strategies, and memory access reduction, have broader applicability. They can be instrumental to enhance the performance of other LiDAR-based algorithms, that range from environmental monitoring to autonomous vehicles, potentially enabling the processing of substantially larger point clouds that were previously intractable, or even real-time processing in low-power embedded devices.

In Chapter 4, we introduced Patterns augmented by Features Epileptic Seizure Detection (PaFESD), a novel strategy for epileptic seizure detection that combines traditional feature extraction techniques with modern pattern recognition methods. We addressed the challenges of manually i) determining feature bounds to exclude invalid segments of EEG signals, and ii) identifying the most representative seizure patterns. Our approach demonstrates the advantages of automatically identifying both the bounds and patterns in a detection algorithm. PaFESD leverages a two-pronged approach: it extracts time-domain and frequency-domain features to filter out non-seizure regions, and employs pattern matching based on Dynamic Time Warping (DTW) operations to identify discriminative seizure patterns. This dual strategy showed substantial improvements in detection accuracy and efficiency, even in scenarios with limited training data. Evaluated on the CHB-MIT database, PaFESD achieved an average $F_1$ score of 98.9%, with perfect detection (100% $F_1$ score) for 20 out of 24 patients, highlighting its potential for reliable health monitoring applications. Furthermore, our method's ability to automatically detect the most seizure/non-seizure discriminative EEG channel suggests the feasibility of developing efficient wearable devices with as few as two electrodes, potentially transforming epilepsy management and significantly improving patients' quality of life and that of their caregivers.

Throughout this thesis, we demonstrated the feasibility and effectiveness of integrating advanced computational techniques to address the challenges of massive sensor-based data processing across different domains. Our findings emphasize the importance of:

- Utilizing heterogeneous computing architectures, such as CPU+GPU systems, to optimize data processing workflows.
- Implementing algorithmic improvements and parallel processing techniques to handle the scale and complexity of modern datasets.
- Relying on up-to-date heterogeneous programming models in order to take advantage of their improved programmability and cross-vendor and cross-architecture portability.
- Leveraging pattern matching models based on the computation of DTW -

Dynamic Time Warping- operations to enhance the accuracy and efficiency of data-driven applications.

As future work, we aim to extend the methodologies developed in this thesis to other sensor-based applications. This includes exploring further optimizations in data processing algorithms, expanding the use of pattern matching techniques based on DTW computations in different contexts, and continuing to improve the integration of these solutions into practical, real-world systems. Additionally, we plan to transfer the lessons learned and results from this thesis into a general-purpose framework, facilitating broader adoption and adaptation in various fields of study related to monitoring applications. We briefly present the follow-up plan in Chapter 6.

## 5.1. Summary of Contributions

The main goal of this thesis has been driven by the necessity to optimize massive data processing in sensor-based applications across various domains by leveraging the advantages of ubiquitous heterogeneous computing architectures. This approach aims to maximize code reusability and minimize development effort. This section summarizes the primary contributions of this thesis, including research publications, project participation, and conference presentations.

### 5.1.1. Previous Research Experience

During a summer stay in the Centro singular de Investigación en Tecnoloxías Intelixentes de la Universidad de Santiago de Compostela (CiTIUS), we got access to a state-of-the-art algorithm in the field of LiDAR data processing, the OWM algorithm explained in Section 3.2.1. The problem of obtaining the seed points to determine the Digital Terrain Model (DTM) in LiDAR point clouds, was solved using the OWM algorithm, but the challenge was to process massive data point clouds in near real-time, with additional restrictions of being able to deploy the algorithm in low-power embedded devices which could be used in drones or autonomous vehicles. This stay was shared with a lot of brilliant researchers, that like us were trying to solve similar problems from different perspectives and fields of study, such as mathematical optimization, computer vision, or Artificial Intelligence, but all focus on the same goal, to improve the performance of algorithms in massive sensor-based data processing.

The foundation for this research was laid by a series of prior investigations

and experiences in the field of heterogeneous computing and massive data processing. These include: 1) An exploration of Shared Virtual Memory (SVM) utilization between Intel CPUs and GPUs integrated within the same system; 2) An internship project focused on real-time reduction of time series data using the Visvalingam-Whyatt algorithm; and 3) Research into new programming models for emerging architectures, particularly oneAPI. These experiences collectively inspired a deeper investigation into heterogeneous computing architectures for optimizing massive data processing across various applications. The following list of research publications and conference presentations encapsulates these formative experiences:

- Felipe Muñoz, José C. Romero, Alejandro Villegas, Ángeles Navarro, Andrés Rodríguez, Rafael Asenjo, *OpenCL 2.0 support for Intel TBB*. XXX Edition of the Conference on Parallelism (JP'19), Cáceres, Sep 18-20, 2019. (National Conference)
- Felipe Muñoz López, Antonio Vilches Reina, Ángeles Navarro, Rafael Asenjo, *Optimisation of the Visvalingam-Whyatt method to reduce time series in real time*. XXXI Edition of the Conference on Parallelism (JP'21/22), Málaga, Sep 22-24, 2021. (National Conference)
- Felipe Muñoz, José C. Romero, Rafael Asenjo, *Using Intel oneAPI to improve the productivity of heterogeneous programming*. XI Winter Seminar CAPAP-H, Universitat Autònoma de Barcelona, Feb 6-7, 2020. (National Conference)

This body of work was the starting point of this thesis, and motivated the steps taken to address the central research question.

### 5.1.2. Stage 1: The LiDAR Data Processing Challenge

The processing of LiDAR data presents a significant challenge due to the vast volume and complexity of the point clouds generated by LiDAR sensors. In Chapter 3, we used as case of study the OWM algorithm and tackled the mentioned challenges by leveraging heterogeneous computing architectures, with a particular focus on optimizations for both CPUs and GPUs. Our objective was to develop efficient algorithms capable of handling the massive datasets produced by LiDAR systems in near real-time, which is critical for applications such as environmental monitoring, urban planning, and autonomous vehicles. The key contribution in this stage was the re-implementation of the OWM algorithm, optimized for GPU architectures, achieving state-of-the-art performance in seed point detection for DTMs.

The results obtained in this stage demonstrated the potential of heterogeneous computing architectures to substantially enhance the efficiency and scalability of LiDAR data processing. Our contributions in this area have established a foundation for further research and development, paving the way for more advanced applications of LiDAR technology in low-power embedded devices.

An early stage of this work was presented at the Conference on High Performance Computing (CHPC'21), and the complete work was subsequently published in the prestigious Journal of Computational Science:

- Felipe Muñoz, Ángeles Navarro, J. Carlos Cabaleiro, Rafael Asenjo, *CPU and GPU oriented optimizations for LiDAR data processing.* Journal of Computational Science, 2023. DOI: 10.1016/j.jocs.2024.102317. (JCR T1/Q2 Journal)
- Jose Carlos Romero, Felipe Muñoz, Antonio Vilches, Andres Rodriguez, Angeles Navarro and Rafael Asenjo *SkyFlow: Heterogeneous Streaming for skyline calculation using FlowGraph and oneAPI.* XXXI Edition of the Conference on Parallelism (JP'21/22), Sep 22-24, 2021, Málaga (Conference Presentation)
- Felipe Muñoz, Rafael Asenjo, Ángeles Navarro, J. Carlos Cabaleiro, *oneAPI implementation of the OWM algorithm for DTM seed points detection.* Conference on High Performance Computing (CHPC'21), 22-27 July 2021. (Conference Presentation)

### 5.1.3. Stage 2: The EEG Seizure Detection Algorithm

Coinciding with the approval of the "CooTSIoT" research project [1] in our research group, which will now found my doctoral studies, my work turns to focus on one of its main objectives: the analysis of EEG signals for automatic detection of epileptic seizures.

In Chapter 4, we focused on the design, development and evaluation of the PaFESD (Patterns augmented by Features Epileptic Seizure Detection) method, which aims to improve the accuracy and efficiency of epileptic seizure detection using electroencephalography (EEG). The key contributions in this stage include:

The advancements made in developing the PaFESD method represent a significant step forward in the field of epileptic seizure detection. The integration of pattern matching with traditional EEG analysis techniques has proven to be

[1] HW-SW co-design and optimization of Time Series based applications for IoT ultra-low power embedded devices `https://cootsiot.github.io/Site/`

a powerful approach, offering improved accuracy and efficiency. Future work in this area could focus on further refining the algorithm, expanding the system to incorporate multimodal data, conducting large-scale clinical trials to validate its effectiveness in real-world settings, as well as implementating the seizure detection on a low-power RISC-V microcontroller to evaluate its real-time performance and energy efficiency on a case of use based on a wearable,.

The results of this research are currently under review for publication in the IEEE Transactions on Biomedical Engineering:

- Felipe Muñoz, Rafael Asenjo, Ángeles Navarro, *PaFESD: Patterns augmented by Features Epileptic Seizure Detection.* Under review, IEEE Transactions on Biomedical Engineering, 2024. (JCR T1/Q2 Journal)

### 5.1.4. Answer to Research Question

The central research question addressed in this thesis was:

> *"How can High Performance Computing (HPC) (high performance programming and high performance architectures) be leveraged to optimize massive sensor-based data processing across different applications?"*

Throughout the chapters, particularly in Chapter 3 and Chapter 4, we showcased the effectiveness of HPC to enhance data processing workflows. By combining algorithmic improvements, parallel processing techniques, and pattern matching models, we achieved substantial performance gains and increased the accuracy of data analysis in both LiDAR and EEG applications. Our research confirms that heterogeneous computing architectures are essential for managing the complexities and demands of massive sensor-based data processing in various fields.

# 6 Future Work

This thesis has explored advanced methodologies and architectural strategies to enhance massive sensor-based data processing, focusing on LiDAR data processing and epileptic seizure detection using EEG. While significant progress has been made, several areas warrant further research to build upon the findings of this work. This chapter outlines potential future directions for research and development.

## 6.1. Advanced Optimizations for LiDAR Data Processing

Future work in the domain of LiDAR data processing should focus on more sophisticated optimization techniques that leverage the full potential of heterogeneous computing architectures. Specifically, the following areas are promising:

- **Real-time Processing:** Enhancing capabilities for real-time processing of LiDAR data by developing low-latency algorithms and optimizing existing ones for real-time applications, such as autonomous navigation and environmental monitoring. Currently, our algorithm takes a complete LiDAR scan as input, processes it, and outputs the initial reference surface needed to obtain the Digital Terrain Model (DTM). As detailed in the experimental results in Section 3.5, our algorithm processes large LiDAR point clouds, significantly larger than those typically used in real-time applications. In a real-time scenario, the algorithm must process a continuous stream of LiDAR data and update the reference surface dynamically. This introduces





additional challenges such as latency constraints and data synchronization. Despite these challenges, we believe our current algorithm provides a robust foundation that can be adapted for real-time processing through appropriate optimizations, achieving excellent performance as demonstrated by our experimental results.

- **Energy-efficient Algorithms:** Designing algorithms that not only improve performance but also reduce energy consumption, particularly for deployment in resource-constrained environments such as drones and mobile platforms. This could involve developing energy-efficient data processing pipelines, optimizing memory access patterns, and minimizing redundant computations. We have already introduced architectures capable of improving the energy efficiency of high-performance GPUs, such as the integrated GPUs (iGPUs) in Intel SoCs described in Section 3.5.3.5. Considering that energy constraints are more stringent in real-time applications on mobile platforms, and that these applications are likely to be less compute-intensive than our current algorithm, we believe that our algorithm can be adapted to run on mobile platforms with the appropriate optimizations.
- **Scalability Studies:** Conducting scalability studies to understand the limitations and performance bottlenecks when processing extremely large datasets. This involves benchmarking on distributed architectures, such as a network of drones or edge devices, to evaluate the scalability of the proposed algorithm. Our current algorithm is designed to process large LiDAR point clouds on a single GPU, which is sufficient for many applications. However, for extremely large datasets, architectures with low memory capacity, or distributed processing scenarios, the algorithm may need to be adapted to handle data partitioning, load balancing, and communication overheads. We can explore innovative approaches such as leveraging edge computing, utilizing hierarchical data processing, and employing advanced load balancing techniques. We believe that our algorithm can be extended to support distributed processing, as LiDAR points preserve spatial information and our Overlap Window Method (OWM) handles Local Lowest Point (LLP) repetition optimally, which are essential for distributed processing.
- **Dynamic Adaptation:** Developing adaptive algorithms that can dynamically adjust their processing strategy based on the available resources, data characteristics, and application requirements. Currently, our algorithm processes LiDAR point clouds with a fixed set of parameters, such as the window size and overlap ratio. By introducing adaptive mechanisms that can learn from the data and optimize these parameters dynamically, we can enhance the efficiency and adaptability of the algorithm across diverse

datasets and environments. We can explore using reinforcement learning to continuously improve the algorithm's performance, leveraging real-time feedback to adjust parameters, and integrating context-aware computing to tailor processing strategies to specific environmental conditions and resource constraints.

- **Data Fusion Techniques:** Investigating data fusion techniques that combine LiDAR data with other sensor modalities such as cameras, radar, and GPS to improve the accuracy and robustness of environmental models. This could involve developing algorithms that integrate data from multiple sources in real-time, addressing challenges related to sensor calibration, data alignment, and synchronization. By leveraging multi-sensor data, we can enhance the capabilities of our OWM algorithm in complex environments, such as urban areas or dense forests.
- **Error Correction and Noise Reduction:** Enhancing algorithms to improve error correction and noise reduction in LiDAR data. This involves developing techniques to filter out noise, correct systematic errors, and interpolate missing data points. By improving the quality of raw LiDAR data, subsequent processing steps can achieve higher accuracy and reliability, which is crucial for applications such as autonomous driving and precision agriculture. In our current work, we do not explicitly address error correction and noise reduction, as our focus is on the initial reference surface extraction. However, these aspects are essential for downstream processing tasks. Future research can explore advanced filtering techniques, incorporated into the OWM algorithm, to enhance the quality of LiDAR data.

## 6.2. Enhanced EEG-based Seizure Detection

In the context of EEG-based seizure detection, several avenues for future research can further advance the effectiveness and reliability of detection systems:

- **Edge Computing Solutions:** Investigating the deployment of EEG-based seizure detection algorithms on edge computing devices to facilitate real-time monitoring and detection in wearable devices, providing immediate alerts to patients and caregivers. PaFESD is designed to output the minimum number of patterns necessary for seizure detection, making it suitable for use in wearable devices. The next step is to design an algorithm that can run on edge devices, such as portable EEG monitors, to enable continuous monitoring and early detection of seizures.

- **Multimodal Data Fusion:** Combining EEG data with other physiological signals, such as ECG (electrocardiogram) and EMG (electromyogram), to enhance the robustness and accuracy of PaFESD. One of the main challenges in EEG-based seizure detection algorithms, particularly those based on machine learning, is the occurrence of interictal oscillations, as described in Section 4.2. By integrating EEG data with other physiological signals, we can improve the detection of these oscillations and reduce false positives.
- **Personalized Detection Systems:** Developing adaptive algorithms that can learn and evolve over time with continuous EEG monitoring data, creating personalized seizure detection systems tailored to individual EEG patterns. As time progresses, a patient's EEG patterns may change or new, unseen patterns may emerge. Our current method has the potential to add new patterns to the batch used for seizure detection, but it lacks a mechanism for continuous updates. Future research can explore the use of online learning techniques to update the detection patterns in real-time, adapting to changes in the patient's EEG signals.
- **New Datasets:** Collecting and curating large-scale EEG datasets from diverse patient populations, including children, adults, and elderly individuals, to evaluate the performance and generalizability of PaFESD across different age groups and conditions. Our algorithm was evaluated on a single dataset [38], and it is essential to test its performance on additional datasets to assess its robustness and reliability.
- **Clinical Validation:** Conducting extensive clinical trials to validate PaFESD in real-world settings and collaborating with medical institutions to gather large-scale EEG datasets for performance assessment across diverse patient populations. Future research can focus on collecting EEG data from various sources, including hospitals, clinics, and research institutions, to create comprehensive datasets for benchmarking and validation.

In summary, the future work outlined here aims to build upon the foundations laid in this thesis, pushing the boundaries of what is possible with heterogeneous computing architectures and advanced data processing methodologies. By exploring these avenues, we can continue to improve the efficiency, accuracy, and applicability of sensor-based data-driven solutions across a wide range of domains.

# Apéndice A
# Resumen en español

Esta tesis aborda los desafíos y oportunidades en el campo de "massive data analytics" mediante el desarrollo de métodos y herramientas novedosas para la extracción y análisis de información, centrándose en la intersección de la teoría de la información, enfoques algorítmicos y consideraciones arquitectónicas en dos aplicaciones específicas basadas en sensores: la extracción de puntos del terreno usando tecnología "Light Detection and Ranging" (LiDAR) y la detección de crisis epilépticas a partir de señales de "Electroencephalography" (EEG). Los objetivos de esta investigación buscan desarrollar métodos y herramientas novedosas para la extracción y análisis de la información de datos de sensores a gran escala, priorizando la precisión, así como la eficiencia computacional y energética en diferentes arquitecturas, buscando desplegar finalmente estas metodologías en dispositivos embebidos para aplicaciones en tiempo real. Entre las plataformas utilizadas se encuentran: "Central Processing Unit" (CPU) multicore, "Graphics Processing Unit" (GPU) integradas y discretas, así como "System on Chip" (SoC) heterogéneos de bajo consumo.

La investigación llevada a cabo en esta tesis se sitúa en el contexto de un aumento sin precedentes en la generación de datos a nivel global, caracterizado principalmente por el volumen, variedad y velocidad. En 2023, se estimó que la producción global de datos alcanzó aproximadamente 120 zettabytes, con previsiones que anticipan un incremento a 180 zettabytes para 2025. Este aumento sin precedentes en la generación de datos, impulsado por los avances en inteligencia artificial, la proliferación del "Internet of Things" (IoT) y la mejora en las comunicaciones por satélite, presenta grandes desafíos a nivel de infraestructura tecnológica, así como oportunidades en diversos sectores. Resulta esencial para el avance tecnológico y la competitividad empresarial el desarrollo de materiales,

dispositivos, métodos y herramientas que nos permitan procesar, analizar y extraer eficientemente conocimiento de estos grandes conjuntos de datos y flujos de información en tiempo real.

En este contexto, nuestra investigación se fundamenta en la necesidad de desarrollar metodologías y algoritmos que aprovechen las nuevas arquitecturas computacionales, capaces de procesar de manera eficiente el creciente volumen y complejidad de los datos, con especial énfasis en aquellos generados por sensores. Las herramientas y métodos convencionales de procesamiento de datos frecuentemente resultan insuficientes para abordar el volumen y flujo de información actuales, particularmente en aplicaciones que requieren procesamiento en tiempo real, abarcando desde sistemas de vehículos autónomos hasta plataformas de monitorización ambiental. Esta realidad nos impulsa a buscar arquitecturas heterogéneas energéticamente eficientes que incorporen CPUs y diversos dispositivos aceleradores, así como la integración eficaz de optimizaciones específicas para cada dispositivo, orientadas a mejorar el flujo computacional y las transacciones de memoria. Para lograr este objetivo, se emplean modelos de programación portables que facilitan a los desarrolladores la explotación de las capacidades inherentes a cada plataforma computacional.

Esta investigación se distingue por su enfoque interdisciplinar, integrando conocimientos de diversas áreas científicas como las ciencias de la computación, la arquitectura de computadoras, las ciencias geoespaciales y la neurociencia. Esta convergencia de disciplinas enriquece la investigación y permite abordar los desafíos del análisis masivo de datos desde múltiples perspectivas. Además, la tesis se caracteriza por su enfoque transversal, incorporando diversas plataformas computacionales. Esta estrategia es fundamental para explotar las oportunidades que presenta el análisis de grandes volúmenes de datos provenientes de sensores. Uno de nuestros objetivos es demostrar cómo la sinergia entre distintas disciplinas científicas y tecnológicas puede conducir a soluciones más eficientes para este complejo escenario.

Como se ha mencionado, un aspecto fundamental de nuestra investigación es el aprovechamiento de las innovaciones tecnológicas, especialmente en lo que respecta a técnicas de procesamiento paralelo, modelos de programación y estrategias de optimización de memoria. En este trabajo se adaptan a las capacidades específicas de cada dispositivo con el fin de abordar de manera eficaz los retos que plantean los entornos de procesamiento de datos a gran escala. Nuestro enfoque trata de desarrollar y mantener plataformas y arquitecturas computacionales adaptadas a las necesidades del volumen y heterogeneidad de las aplicaciones emergentes. Esto se evidencia en las inversiones que están realizando las principales potencias mundiales, como Estados Unidos, China y la Unión Euro-

pea, en investigación sobre semiconductores, inteligencia artificial y computación cuántica.

Al abordar los desafíos del procesamiento masivo de datos basados en sensores, enfatizamos la importancia de la eficiencia energética, que afecta tanto a los dispositivos hardware como a los protocolos de comunicación. Esto es particularmente importante en el contexto de IoT, o en aplicaciones en espacios remotos y aislados, donde el reemplazo de baterías puede ser poco práctico. Esta investigación explora avances en el diseño de modelos computacionales que buscan reducir sustancialmente el consumo de energía, particularmente mediante el uso de sistemas heterogéneos y embebidos con arquitecturas enfocadas en computación paralela, "Single Instruction Multiple Data" (SIMD), y de bajo consumo.

Esta tesis también considera las implicaciones del cambio que se está produciendo en la infraestructura de Tecnologías de la Información, que ahora debe soportar un aumento sin precedentes en aplicaciones y servicios intensivos en datos. Esto incluye la Inteligencia Artificial, el IoT, la realidad virtual, o los gemelos digitales. Este trabajo reconoce la necesidad de soluciones innovadoras y eficientes en plataformas, sistemas y arquitecturas para abordar estos desafíos de manera transversal e integral.

La primera parte de la tesis examina las técnicas de procesamiento asociadas con los datos LiDAR, abordando los desafíos que plantean el volumen y la complejidad de los datos generados. La tecnología LiDAR es una técnica de detección capaz de medir la distancia entre el dispositivo sensor y un objetivo [112], que ha atraído la atención de la comunidad científica, así como de las administraciones públicas y empresas privadas. LiDAR se ha convertido en uno de los principales métodos para la detección de objetos y la captura de la elevación del terreno por ser uno de los métodos con mayor precisión. El registro de la elevación y la topología del terreno, y la capacidad de penetrar a través de la vegetación son algunas de las ventajas sobre métodos tradicionales como la imagen aérea. En las aplicaciones LiDAR en las que se trata de sensar el terreno utilizando vehículos aéreos, el equipo necesario consta de un "Airborne Laser Scanning" (ALS), un "Inertial Measurement Unit" (IMU) y un receptor "Global Positioning System" (GPS). Estos dispositivos se instalan en una aeronave que sobrevuela el terreno obteniendo una nube de puntos georreferenciada. En tierra, un sistema GPS se sincroniza con el receptor GPS de la aeronave. Para cada punto, se registra un conjunto de características: coordenadas $(x, y)$, elevación $(z)$, intensidad, identificación del pulso, número de pulsos y ángulo de escaneo, entre otros. La fotogrametría es otra técnica de detección que obtiene nubes de puntos a partir de imágenes aéreas. La densidad de las nubes de puntos suele ser mayor, ya que se puede configurar por software, pero la precisión es peor que utilizando LiDAR,

particularmente en áreas planas y regiones boscosas, debido a la imposibilidad de penetrar la vegetación y la necesidad de establecer puntos de control en tierra, que son esenciales para proporcionar los parámetros de georreferenciación y disminuir la propagación de errores [36, 112].

El cálculo del "Digital Terrain Model" (DTM) a partir de nubes de puntos ALS es uno de los primeros pasos para obtener información relevante de la superficie escaneada utilizando tecnología LiDAR. Los DTMs se están convirtiendo en productos estándar disponibles en geoportales nacionales y webs de empresas privadas, ya que son esenciales para la elaboración de mapas topográficos. El primer paso para el cálculo del DTM es la extracción de puntos del terreno de la nube de puntos, es decir, la identificación de aquellos puntos en la nube LiDAR que corresponden a la superficie del terreno, discriminando aquellos que corresponden a vegetación, edificaciones, u otro tipo de objetos. Después de esto, se necesita un paso de interpolación para derivar el DTM [112]. En estas primeras etapas de la generación de DTMs, surgen diversos problemas que afectan la precisión y la veracidad de los resultados [16]. Nuestra investigación se centra en el algoritmo "Overlap Window Method", una técnica fundamental para la extracción de puntos del terreno a partir de nubes de puntos ALS. Este método es crucial para la creación de DTMs, ya que trata de extraer con la mayor precisión posible aquellos puntos que forman la superficie inicial de interpolación para el DTM, y que por tanto influenciará a todas las demás etapas del proceso. Se han desarrollado varias implementaciones paralelas optimizadas del algoritmo OWM en C++, SYCL y CUDA, capaces de procesar nubes de puntos de varios millones de puntos en dispositivos comunes, como CPUs y GPUs convencionales, en menos de un segundo. De partida, esto representa una mejora significativa respecto a la implementación original del algoritmo OWM en R [16], que estaba limitada a procesar pequeñas nubes de menos de 50.000 puntos.

Una contribución clave de este trabajo es la exploración de diferentes estructuras de datos para almacenar puntos LiDAR, incluyendo árboles binarios, quadtrees y octrees. Discutimos las ventajas y desventajas entre estas estructuras y aplicamos técnicas heredadas de los algoritmos de ramificación y acotación ("branch-and-bound"), y de algoritmos utilizados en videojuegos "ray tracing" para optimizar la construcción y recorrido de los árboles. Las conclusiones extraídas de los experimentos llevados a cabo utilizando estas estructuras de datos, tienen implicaciones más allá del procesamiento de nubes de puntos LiDAR. Hemos conseguido demostrar cómo la elección de una estructura de datos adecuada puede tener un impacto significativo en el rendimiento del algoritmo OWM, tanto en las etapas de construcción como de recorrido de los árboles.

Nuestro enfoque enfatiza la cuidadosa selección de estructuras de datos adap-

tadas a las características de la microarquitectura de cada acelerador. Dependiendo de la naturaleza de las unidades de procesamiento de cada acelerador, se ha demostrado que existen diferencias significativas en el rendimiento y que es necesario adaptar la implementación del algoritmo OWM a cada arquitectura y en varios niveles: al nivel de la estructura de cada nodo, al nivel de la estructura completa del árbol, y al nivel de la implementación paralela que se ejecute en cada acelerador. Contribuimos con optimizaciones paralelas para arquitecturas CPU y GPU, centrándonos en reducir los requisitos de ancho de banda de memoria, lo cual demostramos que es crucial para la construcción y recorrido en escenarios donde las nubes de puntos LiDAR tienen una densidad elevada de puntos por unidad de superficie. La investigación explora varias estrategias de optimización adaptadas a las capacidades específicas de las diferentes plataformas en nuestro banco de pruebas, incluyendo CPUs multinúcleo, GPUs integradas y GPUs discretas.

En la investigación, hacemos uso de modelos de programación modernos para computación heterogénea, incluyendo OpenMP, "Intel Threading Building Blocks" (TBB), SYCL y CUDA. Nuestro estudio compara estos modelos de programación en términos de rendimiento y portabilidad en diversas plataformas. Encontramos que SYCL, una capa de abstracción software multiplataforma y de código abierto, ofrece un equilibrio prometedor entre la productividad del programador y el rendimiento, permitiendo la ejecución en diferentes dispositivos con un único código fuente sin perdida de rendimiento; incluso es más rápido que CUDA en ciertas plataformas. La evaluación experimental demuestra mejoras significativas de rendimiento sobre las implementaciones base. Por ejemplo, nuestras versiones optimizadas pueden procesar nubes de puntos que contienen más de 40 millones de puntos en menos de un segundo, lo que representa aceleraciones de más de 20 veces en comparación con la implementación secuencial inicial. Proporcionamos análisis detallados del impacto de varias optimizaciones en diferentes arquitecturas, destacando la interacción entre las mejoras algorítmicas y las características específicas del hardware. Este enfoque en la optimización y la adaptabilidad a diferentes arquitecturas nos permite abordar eficazmente los desafíos del procesamiento de datos LiDAR, y de datos provenientes de sensores en general, a gran escala. Nuestros resultados demuestran mejoras significativas en el rendimiento y ofrecen una guía valiosa para saber cómo adaptar algoritmos complejos a diversas plataformas de computación heterogénea.

Los resultados de nuestro trabajo indican que las optimizaciones orientadas a minimizar los accesos a memoria, que hemos denominado como "memoization" (O3), y el ajuste de la granularidad de los nodos hoja de la estructura de datosen función de la densidad de la nube de puntos analizada (O4), tienen un mayor

impacto en el rendimiento de las CPUs. En contraste, las optimizaciones enfocadas en adaptar la estructura de árbol al dispositivo (O1) y explotar todo el paralelismo disponible (O2) muestran mejoras más significativas en las GPUs. Con estas optimizaciones se han conseguido acelerar OWM en un factor de 19x utilizando CPUs y de 83x utilizando GPUs, respecto a la implementación base utilizando OpenMP. También, hemos explorado diferentes estrategias de optimización adaptadas a las capacidades específicas de los dispositivos objetivo en este estudio (CPU, GPU integrada y GPU discreta), así como diferentes mdelos de programación heterogénea (OpenMP, TBB, SYCL, CUDA). Todo esto refuerza nuestras observaciones previas sobre la importancia de adaptar las estrategias de optimización a las características específicas de cada arquitectura. La identificación de estas diferencias en el impacto de las optimizaciones entre CPUs, GPUs, y otras arquitecturas, es crucial para el desarrollo de soluciones eficientes en el procesamiento de datos LiDAR y, por extensión, en el análisis de datos masivos provenientes de sensores. En relación a los entornos de programación, también hemos demostrado que SYCL es muy competitivo (especialmente si se prioriza la portabilidad del código) en comparación con entornos de programación específicos, como TBB (para CPU) o CUDA (para GPU). Para futuras líneas de investigación, contemplamos implementaciones híbridas que utilicen la CPU y la GPU simultáneamente, así como la evaluación de la implementación en dispositivos de bajo consumo que puedan acoplarse con el sensor LiDAR.

La segunda parte de la tesis se centra en la detección de crisis epilépticas mediante el análisis de EEG. La epilepsia es un trastorno crónico del sistema nervioso central que afecta a más de 50 millones de personas en todo el mundo, lo que la convierte en una de las enfermedades neurológicas más comunes a nivel global. Una crisis epiléptica es una alteración repentina y transitoria en la actividad eléctrica del cerebro que puede producir un lapso de atención, alucinaciones, o convulsiones en todo el cuerpo, y aumenta significativamente la probabilidad de que un individuo sufra algún accidente e incluso puede provocarle la muerte de forma súbita. La detección de crisis epilépticas es crucial en el diagnóstico y tratamiento de la epilepsia, y la electroencefalografía es un método no invasivo comúnmente utilizado para este propósito, ya que evalúa la actividad eléctrica producida por las regiones cerebrales desde la superficie del cuero cabelludo, capturada por electrodos y material conductor. El electrocorticograma es el EEG obtenido directamente de la superficie cortical, mientras que el electrograma es el EEG adquirido con electrodos de profundidad. Nuestra investigación se centra únicamente en el EEG registrado desde la superficie del cuero cabelludo. El análisis manual de los datos de EEG puede ser un proceso largo y propenso a errores humanos, lo que puede conducir a retrasos en el diagnóstico y tratamien-

tos subóptimos. En los últimos años, ha habido un creciente interés en el uso de algoritmos de detección automática para mejorar la precisión y eficiencia de la detección de crisis basada en EEG. Es por todo esto que en esta parte de la investigación se aborda la necesidad de mejorar los algoritmos de detección automática en la detección de crisis basada en EEG, con el objetivo de superar las limitaciones del análisis manual. Nuestro trabajo está motivado por el impacto global de la epilepsia y el potencial del EEG como método no invasivo para la detección de crisis, buscando desarrollar soluciones más eficientes y precisas que puedan mejorar significativamente el diagnóstico y tratamiento de esta condición neurológica.

En esta parte, introducimos un nuevo algoritmo, PaFESD ("Pattern augmented by Features Epileptic Seizure Detection"), que combina el uso de la distancia "Dynamic Time Warping" (DTW) para la extracción de patrones característicos en las crisis epilépticas, con el análisis de señales basado en "features", con el fin de mejorar la precisión en la identificación de crisis. Hasta donde hemos podido investigar, *PaFESD*, es el primero en fusionar estas dos estrategias mencionadas: el uso de DTW para encontrar automáticamente patrones que discriminen las crisis de las regiones sin crisis en un canal de EEG, también conocido como "pattern matching", combinado con el análisis de características de la señal para filtrar regiones sin crisis que podrían confundir la fase de coincidencia de patrones. Además, contribuimos con una nueva métrica de calidad de detección que denominamos "Discrimination Ratio" (DR), diseñada para buscar patrones de crisis que no produzcan falsos positivos (falsas alarmas). Demostramos que nuestro método puede encontrar automáticamente patrones discriminativos de crisis incluso cuando utilizamos solo alrededor del 22% de la señal EEG (incluyendo como máximo 4 crisis) para entrenar nuestro modelo. Asimismo, aprovechamos técnicas de filtrado de señales y eliminación de artefactos para reducir el ruido debido a movimientos musculares o interferencias eléctricas. Finalmente, nuestra propuesta detecta automáticamente el canal más discriminativo, de modo que, una vez identificados los patrones de crisis en ese canal, pueden utilizarse para un algoritmo de detección de crisis que se ejecute en un dispositivo conectado a solo dos electrodos (en gafas, diadema, auriculares o similar). Esto es especialmente interesante para aplicaciones de monitorización continua y en tiempo real, ya que evita el estigma que conlleva el uso de otro tipo de sensores y métodos invasivos.

Para desarrollar y validar nuestro enfoque, utilizamos la base de datos CHB-MIT, una colección de grabaciones EEG de 24 pacientes con epilepsia. Este conjunto de datos, que comprende 664 archivos .edf con un total de 198 crisis y más de 980 horas de grbaciones, proporciona una base sólida para nuestra investigación. La elección de esta base de datos es estratégica, ya que nos permite

trabajar con un conjunto de datos diverso y representativo de la variabilidad en las manifestaciones de crisis epilépticas entre diferentes individuos, ya que i) las crisis de cada paciente son de diferente naturaleza, ii) las crisis de un mismo paciente tampoco suelen ocurrir en el mismo lugar del cerebro ("lateralization"), y iii) las crisis de diferentes pacientes tampoco tienen las mismas características. Esto es crucial para desarrollar un método de detección que sea robusto y generalizable, y que pueda funcionar satisfactoriamente en un amplio grupo de pacientes.

En resumen, la implementación de PaFESD implica los siguientes pasos:

- Preprocesamiento de la señal: Aplicamos técnicas de filtrado de señales y eliminación de artefactos utilizando geometría "Riemannian" para reducir el ruido y mejorar la calidad de la señal.
- "Pattern Matching": Utilizamos DTW para comparar posibles patrones de crisis con épocas en la señal, optimizado mediante un vector "Do-Not-Compute" (DNC) que filtra epochs irrelevantes. Discriminamos epochs utilizando cinco características extraídas de cada epoch: amplitud pico a pico, energía y densidad espectral de potencia en tres rangos de frecuencia (2.5-12Hz, 12-18Hz y 18-35Hz).
- Cálculo de DR: Esta métrica ayuda a identificar los patrones de crisis más discriminativos mientras minimiza el número de falsos positivos.
- Clustering: Seleccionamos la mejor solución (el par patrón-característica) utilizando el centro de masas de las soluciones obtenidas del entrenamiento (política "centroid").

Nuestra evaluación experimental demuestra mejoras significativas sobre los métodos existentes, particularmente en términos de sensibilidad y tasas de falsas alarmas.

El enfoque PaFESD muestra un gran potencial en la detección de una amplia gama de tipos de crisis, manteniendo al mismo tiempo una alta precisión y eficiencia. PaFESD logra una detección perfecta ($F_1 = 1$, sin falsos positivos ni falsos negativos) para 20 de los 24 pacientes, lo que representa el 83% del conjunto de datos. Un estudio adicional utilizando la política denominada "Oracle" demuestra que, mejorando la técnica de "clustering" para selección del mejor entrenamiento, podríamos mejorar aún más los resultados, alcanzando una precisión del 99.5%, una sensibilidad del 100% y un $F_1$ del 99.7%. La investigación también considera las implicaciones de estos resultados para "edge computing" y el procesamiento de datos en tiempo real. Discutimos los trabajos en curso para desplegar PaFESD en dispositivos embebidos como Raspberry Pis y microcontroladores RISC-V, así como la aceleración de etapas del algoritmo utilizando aceleradores FPGA y GPU.

A modo de síntesis, cabe destacar que esta tesis se fundamente en una perspectiva interdisciplinar y en el aprovechamiento de los avances tecnológicos en arquitecturas de computación, "Application Programming Interface" (APIs) para computación heterogénea y técnicas de procesamiento paralelo. Los objetivos abarcan el avance de las técnicas de análisis de datos LiDAR, la mejora de metodologías de detección de crisis epilépticas, contribuyendo tanto a aplicaciones como a marcos teóricos en el campo del análisis masivo de datos obtenidos de sensores. Nuestro trabajo demuestra la viabilidad y eficacia de integrar técnicas computacionales avanzadas para abordar los desafíos del procesamiento masivo de datos en diferentes dominios. Los resultados obtenidos enfatizan la importancia de un enfoque integral: utilizar arquitecturas de computación heterogéneas, implementar mejoras algorítmicas y técnicas de procesamiento paralelo, y confiar en modelos de programación heterogéneos actualizados.

El trabajo futuro se centrará en optimizar aún más estas metodologías y extenderlas a otras aplicaciones basadas en sensores. Para el procesamiento LiDAR, esto incluye desarrollar capacidades de procesamiento en tiempo real, diseñar algoritmos energéticamente eficientes para entornos con recursos limitados, realizar estudios de escalabilidad para conjuntos de datos extremadamente grandes donde las nubes de puntos no pueden ser completamente volcadas en memoria, y explorar técnicas de adaptación dinámica a estas condiciones. En la detección de crisis epilépticas basada en EEG, las direcciones futuras incluyen investigar soluciones de "edge computing" para monitorización en tiempo real, integrar la fusión de datos multimodales para mejorar la precisión de la detección, desarrollar sistemas de detección personalizados y realizar validaciones clínicas extensas.

Con esta investigación esperamos contribuir a aprovechar todo el potencial de las nuevas arquitectureas enfocadas al análisis masivo de datos. De la misma manera, aspiramos a mejorar las capacidades y aplicaciones de los sistemas de toma de decisiones basados en datos, fomentando a seguir ampliando con esta las fronteras de nuestro conocimiento y mejorando las capacidades de los sistemas computacionales para el análisis de datos masivos.

# Bibliography


[1] Dennis Abts, Garrin Kimmell, Andrew Ling, John Kim, Matt Boyd, Andrew Bitar, Sahil Parmar, Ibrahim Ahmed, Roberto DiCecco, David Han, John Thompson, Michael Bye, Jennifer Hwang, Jeremy Fowers, Peter Lillian, Ashwin Murthy, Elyas Mehtabuddin, Chetan Tekur, Thomas Sohmers, Kris Kang, Stephen Maresh, and Jonathan Ross. A software-defined tensor streaming multiprocessor for large-scale machine learning. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 567–580, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450386104. doi: 10.1145/3470496.3527405. URL `https://doi.org/10.1145/3470496.3527405`.

[2] Akeel Al-Sakaa et al. Effective electroencephalogram based epileptic seizure detection using support vector machine and statistical moment's features. *Int. J. of Electrical and Computer Eng. (IJECE)*, 2022. doi: 10.11591/ijece.v12i5.pp5204-5213.

[3] Emina Alickovic, Jasmin Kevric, and Abdulhamit Subasi. Performance evaluation of empirical mode decomposition, discrete wavelet transform, and wavelet packed decomposition for automated epileptic seizure detection and prediction. *Biomedical Signal Processing and Control*, 39, 2018. ISSN 1746-8094.

[4] Allied Market Research. Data center chip market. `https://www.alliedmarketresearch.com/data-center-chip-market`, 2023. URL `https://www.alliedmarketresearch.com/data-center-chip-market`. Accessed: [Insert access date].

[5] Mina Amiri, Birgit Frauscher, and Jean Gotman. Interictal coupling of hfos and slow oscillations predicts the seizure-onset pattern in mesiotemporal lobe epilepsy. *Epilepsia*, 60(6):1160–1170, 2019. doi: https://doi.org/10.

1111/epi.15541. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/epi.15541.

[6] AnandTech. Intel Goes Full XPU: Falcon Shores to Combine x86 and Xe for Supercomputers. https://www.anandtech.com/show/17268/intel-goes-full-xpu-falcon-shores-to-combine-x86-and-xe-for-supercomputers, 2022. Accessed: 2024-02-27.

[7] AnandTech. Apple Announces M3 SoC Family: M3, M3 Pro, and M3 Max Make Their Marks. https://www.anandtech.com/show/21116/apple-announces-m3-soc-family-m3-m3-pro-and-m3-max-make-their-marks, 2024. Accessed: 2024-02-27.

[8] Saleh Baghersalimi et al. Personalized real-time federated learning for epileptic seizure detection. *IEEE J. of Biomedical and Health Informatics*, 26(2):898–909, 2022. doi: 10.1109/JBHI.2021.3096127.

[9] Alexandre Barachant, Quentin Barthélemy, Jean-Rémi King, Alexandre Gramfort, Sylvain Chevallier, Pedro L. C. Rodrigues, Emanuele Olivetti, Vladislav Goncharenko, Gabriel Wagner vom Berg, Ghiles Reguig, Arthur Lebeurrier, Erik Bjäreholt, Maria Sayu Yamamoto, Pierre Clisson, and Marie-Constance Corsi. pyriemann/pyriemann: v0.5, june 2023. URL https://doi.org/10.5281/zenodo.8059038.

[10] Quentin Barthélemy, Louis Mayaud, David Ojeda, and Marco Congedo. The riemannian potato field: A tool for online signal quality index of EEG. *IEEE T. on Neural Systems and Rehabilitation Eng.*, 2019.

[11] Jan Beitner. Lipo. https://github.com/jdb78/lipo, 2021.

[12] Sándor Beniczky, Samuel Wiebe, Jesper Jeppesen, William O Tatum, Milan Brazdil, Yuping Wang, Susan T Herman, and Philippe Ryvlin. Automated seizure detection using wearable devices: A clinical practice guideline of the international league against epilepsy and the international federation of clinical neurophysiology. *Epilepsia*, 62, 2021. doi: 10.1111/epi.16818.

[13] Marshall Bern, David Eppstein, and Shang-Hua Teng. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry & Applications*, 09(06):517–532, 1999.

[14] Sarah Blum, N S J Jacobsen, Martin G. Bleichner, and Stefan Debener. A riemannian modification of artifact subspace reconstruction for eeg artifact handling. *Frontiers in Human Neuroscience*, 13, 2019.

[15] Elisa Bruno, Sebastian Böttcher, Andrea Biondi, Nino Epitashvili, Nikolay V Manyakov, Simon Lees, Andreas Schulze-Bonhage, and Mark P Richardson. Post-ictal accelerometer silence as a marker of post-ictal immobility. *Epilepsia*, 61:1397 – 1405, 2020. doi: 10.1111/epi.16552. URL `http://doi.org/10.1111/epi.16552`.

[16] Sandra Buján, Miguel Cordero, and David Miranda. Hybrid overlap filter for lidar point clouds using free software. *Remote Sensing*, 12:1051, 03 2020. doi: 10.3390/rs12071051.

[17] Duo Chen, Suiren Wan, Jing Xiang, and Forrest Sheng Bao. A high-performance seizure detection algorithm based on discrete wavelet transform (DWT) and EEG. *PLOS ONE*, 12(3):1–21, 03 2017. doi: 10.1371/journal.pone.0173138.

[18] Ziyue Chen, Bingbo Gao, and Bernard Devereux. State-of-the-art: DTM generation using airborne LIDAR data. *Sensors*, 17(1), 2017. doi: 10.3390/s17010150.

[19] Thomas Dalgaty, Filippo Moro, Yiğit Demirağ, Alessio De Pra, Giacomo Indiveri, Elisa Vianello, and Melika Payvand. Mosaic: in-memory computing and routing for small-world spike-based neuromorphic systems. *Nature Communications*, 15(1):142, January 2024. ISSN 2041-1723. doi: 10.1038/s41467-023-44365-x. URL `https://doi.org/10.1038/s41467-023-44365-x`.

[20] Hisham Daoud and Magdy A. Bayoumi. Efficient epileptic seizure prediction based on deep learning. *IEEE T. on Biomedical Circuits and Systems*, 13(5):804–813, 2019. doi: 10.1109/TBCAS.2019.2929053.

[21] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974. doi: 10.1109/JSSC.1974.1050511.

[22] Hui Ding et al. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1 (2):1542–1552, aug 2008. ISSN 2150-8097. doi: 10.14778/1454159.1454226.

[23] Ed Sperling. The Rising Price of Power in Chips. `https://semiengineering.com/the-rising-price-of-power-in-chips/`. Accessed: 2024-02-27.

[24] Edge AI and Vision Alliance. CUDA Refresher: Reviewing the Origins of GPU Computing. https://www.edge-ai-vision.com/2020/06/cuda-refresher-reviewing-the-origins-of-gpu-computing/, June 2020. Accessed: 2024-02-27.

[25] Learning EEG. 10-20 system, 2020. URL https://www.learningeeg.com/montages-and-technical-components.

[26] Ali Emami et al. Seizure detection by convolutional neural network-based analysis of scalp electroencephalography plot images. *NeuroImage: Clinical*, 22, 2019. ISSN 2213-1582. doi: https://doi.org/10.1016/j.nicl.2019.101684.

[27] Ericsson. Future Network Requirements for XR Apps. https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/future-network-requirements-for-xr-apps, 2023. Accessed: 2024-02-27.

[28] European Commission. AI innovation package to support Artificial Intelligence startups and SMEs. https://ec.europa.eu/commission/presscorner/detail/en/ip_24_383, 2024. Accessed: 2024-02-27.

[29] Expert Market Research. Global Big Data Market Outlook. https://www.expertmarketresearch.com/reports/big-data-market, 2023. Accessed: 2024-02-27.

[30] Wu-chun Feng. Making a case for efficient supercomputing: It is time for the computing community to use alternative metrics for evaluating performance. *Queue*, 1(7):54–64, oct 2003. ISSN 1542-7730. doi: 10.1145/957717.957772. URL https://doi.org/10.1145/957717.957772.

[31] Paul Feyerabend. *Against Method*. New Left Books, London, 1988.

[32] Hans Fischer. *A History of the Central Limit Theorem: From Classical to Modern Probability Theory*. Sources and Studies in the History of Mathematics and Physical Sciences. Springer, New York, NY, 2011. ISBN 978-0-387-87857-7. doi: 10.1007/978-0-387-87857-7.

[33] Fortune. Intel, TSMC, and Samsung are deamnding Biden double the funds on hand for chips. https://fortune.com/2024/02/26/gina-raimondo-chips-act-intel-tsmc-samsung-70-billion/, February 2024. Accessed: 2024-02-27.

[34] V.A. Fursov, Ye.V. Goshin, and A.P. Kotov. The hybrid CPU/GPU implementation of the computational procedure for digital terrain models generation from satellite images. *Computer Optics*, 40(5):721–729, 2016. doi: 10.3390/rs70810996.

[35] Kirill Garanzha, Jacopo Pantaleoni, and David McAllister. Simpler and faster hlbvh with work queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG'11, page 59–64, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450308960. doi: 10.1145/2018323.2018333. URL https://doi.org/10.1145/2018323.2018333.

[36] Alejandro Lorenzo Gil, Laia Núñez-Casillas, Martin Isenburg, Alfonso Alonso Benito, José Julio Rodrigo Bello, and Manuel Arbelo. A comparison between lidar and photogrammetry digital terrain models in a forest area on tenerife island. *Canadian Journal of Remote Sensing*, 39(5): 396–409, 2013.

[37] Toni Giorgino. Computing and visualizing dynamic time warping alignments in R: The dtw package. *J. of Statistical Software*, 2009. doi: 10.18637/jss.v031.i07.

[38] Glass L Goldberger AL, Amaral LAN. CHB-MIT scalp EEG database, 2010. URL https://physionet.org/content/chbmit.

[39] Dasa Gorjan, Klaus Gramann, Kevin De Pauw, and Uros Marusic. Removal of movement-induced EEG artifacts: current state of the art and guidelines. *J. of Neural Engineering*, 19(1), feb 2022. doi: 10.1088/1741-2552/ac542c.

[40] Alexandre Gramfort et al. MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience*, 7(267), 2013. doi: 10.3389/fnins.2013.00267.

[41] GSMA. The mobile economy 2023. https://www.gsma.com/mobileeconomy/wp-content/uploads/2023/03/270223-The-Mobile-Economy-2023.pdf, 2023. Accessed: 2024-02-27.

[42] Kathleen E. Hamilton, Catherine D. Schuman, Steven R. Young, Ryan S. Bennink, Neena Imam, and Travis S. Humble. Accelerating scientific computing in the post-moore's era. *ACM Trans. Parallel Comput.*, 7 (1), mar 2020. ISSN 2329-4949. doi: 10.1145/3380940. URL https://doi.org/10.1145/3380940.

[43] Dr. Varsha K. Harpale and Vinayak K. Bairagi. Time and frequency domain analysis of EEG signals for seizure detection: A review. *2016 Int. Conf. on Microelectronics, Computing and Comm. (MicroCom)*, pages 1–6, 2016.

[44] Matthieu Herrmann and Geoffrey I. Webb. Early abandoning and pruning for elastic distances including dynamic time warping. *Data Mining and Knowledge Discovery*, 35(6):2577–2601, Nov 2021. doi: 10.1007/s10618-021-00782-4.

[45] Anson Ho, Ege Erdil, and Tamay Besiroglu. Limits to the energy efficiency of cmos microprocessors, 2023.

[46] Marius Hobbhahn and Tamay Besiroglu. Predicting gpu performance, 2022. URL `https://epochai.org/blog/predicting-gpu-performance`. Accessed: 2024-04-20.

[47] Marius Hobbhahn, Lennart Heim, and Gökçe Aydos. Trends in machine learning hardware, 2023. URL `https://epochai.org/blog/trends-in-machine-learning-hardware`. Accessed: 2024-04-20.

[48] Rashina Hoda, Norsaremah Salleh, and John Grundy. The rise and evolution of agile software development. *IEEE Software*, 35(5):58–63, 2018. doi: 10.1109/MS.2018.290111318.

[49] R. Hopfengärtner et al. An efficient, robust and fast method for the offline detection of epileptic seizures in long-term scalp EEG recordings. *Clinical Neurophysiology*, 118(11), 2007. ISSN 1388-2457. doi: https://doi.org/10.1016/j.clinph.2007.07.017.

[50] Rüdiger Hopfengärtner et al. Automatic seizure detection in long-term scalp EEG using an adaptive thresholding technique: A validation study for clinical routine. *Clinical Neurophysiology*, 125(7), 2014. ISSN 1388-2457. doi: https://doi.org/10.1016/j.clinph.2013.12.104.

[51] M. Shamim Hossain et al. Applying deep learning for epilepsy seizure detection and brain mapping visualization. *ACM Trans. Multimedia Comput. Commun. Appl.*, 15(1s), feb 2019. ISSN 1551-6857. doi: 10.1145/3241056.

[52] Seyed Hossein Mortazavi, Mohammad Salehe, Moshe Gabel, and Eyal de Lara. Data management systems for the hierarchical edge. *GetMobile: Mobile Comp. and Comm.*, 27(2):11–17, aug 2023. ISSN 2375-0529. doi: 10.1145/3614214.3614218. URL `https://doi.org/10.1145/3614214.3614218`.

[53] Alexander E. Hramov, Vladimir A. Maksimenko, and Alexander N. Pisarchik. Physical principles of brain–computer interfaces and their applications for rehabilitation, robotics and control of human brain states. *Physics Reports*, 918:1–133, 2021. ISSN 0370-1573. doi: https://doi.org/10.1016/

j.physrep.2021.03.002. URL https://www.sciencedirect.com/science/article/pii/S0370157321001095. Physical principles of brain–computer interfaces and their applications for rehabilitation, robotics and control of human brain states.

[54] Xiangyun Hu, Xiaokai Li, and Yongjun Zhang. Fast filtering of lidar point cloud in urban areas based on scan line segmentation and gpu acceleration. *IEEE Geoscience and Remote Sensing Letters*, 10(2):308–312, 2013. doi: 10.1109/LGRS.2012.2205130.

[55] Xiangyun Hu, Lizhi Ye, Shiyan Pang, and Jie Shan. Semi-Global filtering of airborne LiDAR data for fast extraction of digital terrain models. *Remote Sensing*, 7:10996–11015, 2015. doi: 10.3390/rs70810996.

[56] IEEE. 2022 International Roadmap for Devices and Systems (IRDS) White Paper on More than Moore (MtM). https://irds.ieee.org/images/files/pdf/2022/2022IRDS_WP-MtM.pdf, 2022. Accessed: 2024-02-27.

[57] IEEE Computer Society. Margaret Hamilton: First Software Engineer. https://www.computer.org/publications/tech-news/events/what-to-know-about-the-scientist-who-invented-the-term-software-engineering, 2020. Accessed: 2024-02-27.

[58] Intel. Porting Guide for ICC Users to DPC++ or ICX. https://www.intel.com/content/www/us/en/developer/articles/guide/porting-guide-for-icc-users-to-dpcpp-or-icx.html. Accessed: 2024-02-27.

[59] Intel Corporation. oneAPI Specification 1.2 Rev. 1. https://spec.oneapi.io/versions/1.2-rev-1/, 2023. Accessed: 2024-03-03.

[60] Intel Corporation. Intel LLVM Technology - SYCL Branch. https://github.com/intel/llvm/tree/sycl, 2024. Accessed: 2024-02-27.

[61] International Data Corporation (IDC). Worldwide Semiannual Big Data and Analytics Software. https://www.idc.com/getdoc.jsp?containerId=prEUR250058223, 2023. Accessed: 2024-02-27.

[62] International Data Corporation (IDC). Understanding XaaS Adoption, Key Customer Motivations. https://www.cloudblue.com/wp-content/uploads/2023/03/idc-understanding-xaas.pdf?utm_source=linkedin&utm_medium=post&utm_campaign=701Do000000gX6YIAU, 2023. Accessed: 2024-02-27.

[63] International Data Corporation (IDC). Top 10 Worldwide IT Industry 2024 Predictions: Mastering AI Everywhere, November 2023. URL https://blogs.idc.com/2023/11/01/top-10-worldwide-it-industry-2024-predictions-mastering-ai-everywhere/. Accessed: 2024-02-27.

[64] International Data Corporation (IDC). IDC FutureScape: Worldwide IT Industry 2024 Predictions. Technical report, International Data Corporation (IDC), 2023. URL https://www.idc.com/getdoc.jsp?containerId=US50435423. Accessed: 2024-02-27.

[65] International Data Corporation (IDC) and Seagate. The Digitization of the World From Edge to Core. Technical report, Seagate Technology LLC, 2018. URL https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf. Accessed: 2024-02-27.

[66] Intersect360 Research. Worldwide HPC and AI 2022 Total Market Size and 2023-2027 Forecast: Vertical Markets. Technical report, Intersect360 Research, 2023. URL https://www.intersect360.com/report/worldwide-hpc-and-ai-2022-total-market-size-and-2023-2027-forecast-vertical-markets/. Accessed: 2024-02-27.

[67] IoT Analytics. State of IoT 2023: Number of connected IoT devices growing 16% to 16.0 billion globally. Technical report, IoT Analytics, May 2023. URL https://iot-analytics.com/wp/wp-content/uploads/2023/05/Insights-Release-State-of-IoT-2023-Number-of-connected-IoT-devices-growing-16-to-16.0-billion-globally.pdf. Accessed: 2024-02-27.

[68] Thomas Jackson and Ian Richard Hodgkinson. Is there a role for knowledge management in saving the planet from too much data? *Knowledge Management Research & Practice*, 21(3):427–435, 2023. doi: 10.1080/14778238.2023.2192580. URL https://doi.org/10.1080/14778238.2023.2192580.

[69] Dongni Johansson, Fredrik Ohlsson, David Krýsl, Bertil Rydenhag, Madeleine Czarnecki, Niclas Gustafsson, Jan Wipenmyr, Tomas McKelvey, and Kristina Malmgren. Tonic-clonic seizure detection using accelerometry-based wearable sensors: A prospective, video-eeg controlled study. *Seizure*, 65:48–54, 2019. doi: 10.1016/j.seizure.2018.12.024. URL http://doi.org/10.1016/j.seizure.2018.12.024.

[70] Ji-Hoon Kang, Heechang Shin, Ki Seok Kim, Min-Kyu Song, Doyoon Lee, Yuan Meng, Chanyeol Choi, Jun Min Suh, Beom Jin Kim, Hyunseok Kim, Anh Tuan Hoang, Bo-In Park, Guanyu Zhou, Suresh Sundaram, Phuong Vuong, Jiho Shin, Jinyeong Choe, Zhihao Xu, Rehan Younas, Justin S. Kim, Sangmoon Han, Sangho Lee, Sun Ok Kim, Beomseok Kang, Seungju Seo, Hyojung Ahn, Seunghwan Seo, Kate Reidy, Eugene Park, Sungchul Mun, Min-Chul Park, Suyoun Lee, Hyung-Jun Kim, Hyun S. Kum, Peng Lin, Christopher Hinkle, Abdallah Ougazzaden, Jong-Hyun Ahn, Jeehwan Kim, and Sang-Hoon Bae. Monolithic 3D integration of 2D materials-based electronics towards ultimate edge computing solutions. *Nature Materials*, 22(12):1470–1477, December 2023. ISSN 1476-1122, 1476-4660. doi: 10.1038/s41563-023-01704-z. URL `https://www.nature.com/articles/s41563-023-01704-z`.

[71] Tero Karras. Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, page 33–37, Goslar, DEU, 2012. Eurographics Association. ISBN 9783905674415.

[72] Tero Karras and Timo Aila. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, HPG '13, page 89–99, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321358. doi: 10.1145/2492045.2492055. URL `https://doi.org/10.1145/2492045.2492055`.

[73] Yinan Ke, Mulya Agung, and Hiroyuki Takizawa. neosycl: a sycl implementation for sx-aurora tsubasa. In *The International Conference on High Performance Computing in Asia-Pacific Region*, HPCAsia '21, page 50–57, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450388429. doi: 10.1145/3432261.3432268. URL `https://doi.org/10.1145/3432261.3432268`.

[74] Taeho Kim et al. Epileptic seizure detection and experimental treatment: A review. *Frontiers in Neurology*, 11, 2020. ISSN 1664-2295. doi: 10.3389/fneur.2020.00701.

[75] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[76] Davis E. King. dlib find_max_global function, 2020. URL `http://dlib.net/python/index.html#dlib.find_max_global`.

[77] Marius Klug and Klaus Gramann. Identifying key factors for improving ICA-based decomposition of EEG data in mobile and stationary experiments. *European J. of Neuroscience*, 54(12), 2021. doi: https://doi.org/10.1111/ejn.14992.

[78] David C. Krakauer, editor. *Foundational Papers in Complexity Science*, volume I. The SFI Press Scholars Series, 2024. ISBN 978-1-947864-56-6. Paperback: $24.99, Hardcover: $39.99.

[79] Shitanshu Kusmakar, Chandan K. Karmakar, Bernard Yan, Terence J.O'Brien, Ramanathan Muthuganapathy, and Marimuthu Palaniswami. Automated detection of convulsive seizures using a wearable accelerometer device. *IEEE Transactions on Biomedical Engineering*, 66:421–432, 2019. doi: 10.1109/TBME.2018.2845865. URL `http://doi.org/10.1109/TBME.2018.2845865`.

[80] Nanxi Lai, Zhisheng Li, Cenglin Xu, Yi Wang, and Zhong Chen. Diverse nature of interictal oscillations: Eeg-based biomarkers in epilepsy. *Neurobiology of Disease*, 177:105999, 2023. ISSN 0969-9961. doi: https://doi.org/10.1016/j.nbd.2023.105999. URL `https://www.sciencedirect.com/science/article/pii/S096999612300013X`.

[81] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH Construction on GPUs. *Computer Graphics Forum*, 2009. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2009.01377.x.

[82] Yang Li et al. Epileptic seizure detection in EEG signals using a unified temporal-spectral squeeze-and-excitation network. *IEEE T. on Neural Systems and Rehabilitation Engineering*, 28(4), 2020. doi: 10.1109/TNSRE.2020.2973434.

[83] Xian Liu, Dawei Lu, Aiqian Zhang, Qian Liu, and Guibin Jiang. Data-driven machine learning in environmental pollution: Gains and problems. *Environmental Science & Technology*, 56(4):2124–2133, 2022. doi: 10.1021/acs.est.1c06157.

[84] Lojini Logesparan, Esther Rodríguez-Villegas, and Alexander J. Casson. The impact of signal normalization on seizure detection using line length features. *Medical & Biological Engineering & Computing*, 53, 2015.

[85] Giuseppe Loseto, Floriano Scioscia, Michele Ruta, Filippo Gramegna, Saverio Ieva, Corrado Fasciano, Ivano Bilenchi, Davide Loconte, and Eugenio Di Sciascio. A cloud-edge artificial intelligence framework for sensor networks.

In *2023 9th International Workshop on Advances in Sensors and Interfaces (IWASI)*, pages 149–154, 2023. doi: 10.1109/IWASI58316.2023.10164335.

[86] Brian Nils Lundstrom, Benjamin H Brinkmann, and Gregory A Worrell. Low frequency novel interictal eeg biomarker for localizing seizures and predicting outcomes. *Brain Communications*, 3(4):fcab231, 2021.

[87] Benno Mahler, Sofia Carlsson, Tomas Andersson, and Torbjörn Tomson. Risk for injuries and accidents in epilepsy. *Neurology*, 90:e779 – e789, 2018.

[88] Nadia Mammone, Fabio La Foresta, and Francesco Carlo Morabito. Automatic artifact rejection from multichannel scalp eeg by wavelet ica. *IEEE Sensors Journal*, 12:533–542, 2012.

[89] Malik Muhammad Naeem Mannan, Muhammad Ahmad Kamran, and Myung Yung Jeong. Identification and removal of physiological artifacts from electroencephalogram signals: A review. *IEEE Access*, 6, 2018. doi: 10.1109/ACCESS.2018.2842082.

[90] Dan C. Marinescu. Chapter 3 - parallel processing and distributed computing. In Dan C. Marinescu, editor, *Cloud Computing (Third Edition)*, pages 41–94. Morgan Kaufmann, third edition edition, 2023. ISBN 978-0-323-85277-7. doi: https://doi.org/10.1016/B978-0-32-385277-7.00010-5. URL `https://www.sciencedirect.com/science/article/pii/B9780323852777000105`.

[91] Bernard Marr. 2024 IoT and Smart Device Trends: What You Need to Know for the Future. *Forbes*, October 2023. URL `https://www.forbes.com/sites/bernardmarr/2023/10/19/2024-iot-and-smart-device-trends-what-you-need-to-know-for-the-future/`. Accessed: 2024-02-27.

[92] Xuelian Meng, Nate Currit, and Kaiguang Zhao. Ground filtering algorithms for airborne lidar data: A review of critical issues. *Remote Sensing*, 2(3):933–860, 2010.

[93] Maher Moakher. A differential geometric approach to the geometric mean of symmetric positive-definite matrices. *SIAM J. on Matrix Analysis and Applications*, 26(3):735–747, 2005. doi: 10.1137/S0895479803436937.

[94] Shady M. K. Mohamed et al. Towards automated quality assessment measure for EEG signals. *Neurocomputing*, 2017.

[95] G.E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998. doi: 10.1109/JPROC.1998.658762.

[96] Gordon E. Moore. Progress in digital integrated electronics [technical literaiture, copyright 1975 ieee. reprinted, with permission. technical digest. international electron devices meeting, ieee, 1975, pp. 11-13.]. *IEEE Solid-State Circuits Society Newsletter*, 11(3):36–37, 2006. doi: 10.1109/N-SSC.2006.4804410.

[97] Takashi Morishita et al. Epilepsy detection based on riemann potato in noisy environment. *Applied Bionics and Biomechanics*, 2022. doi: 10.1155/2022/8311249.

[98] Ashish Pandharipande, Chih-Hong Cheng, Justin Dauwels, Sevgi Z. Gurbuz, Javier Ibanez-Guzman, Guofa Li, Andrea Piazzoni, Pu Wang, and Avik Santra. Sensing and machine learning for automotive perception: A review. *IEEE Sensors Journal*, 23(11):11097–11115, 2023. doi: 10.1109/JSEN.2023.3262134.

[99] Marco A. Pinto-Orellana and Fábio R. Cerqueira. Patient-dependent epilepsy seizure detection using random forest classification over one-dimension transformed EEG data. *bioRxiv*, 2016. doi: 10.1101/070300.

[100] J. Prasanna, Subathra, et al. Automated epileptic seizure detection in pediatric subjects of CHB-MIT EEG database—a survey. *Journal of Personalized Medicine*, 11(10), 2021. ISSN 2075-4426. doi: 10.3390/jpm11101028.

[101] S. Pravin Kumar et al. Entropies based detection of epileptic seizures with artificial neural network classifiers. *Expert Systems with Applications*, 37 (4):3284–3291, 2010. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2009.09.051.

[102] Qualcomm. Snapdragon W5+ Gen 1 Wearable Platform. `https://www.qualcomm.com/products/mobile/snapdragon/wearables/snapdragon-w5-plus-gen-1-wearable-platform#Overview`, 2024. Accessed: 2024-02-27.

[103] James Reinders, Ben Ashbaugh, James Broadman, Michael Kinsner, John Pennycook, and Xinmin Tian. *Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL*. Apress, 2021.

[104] Alejandro Pasos Ruiz et al. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2), 2021. doi: 10.1007/s10618-020-00727-3.

[105] Karl Rupp. 50 years of microprocessor trend data, 2022. URL `https://github.com/karlrupp/microprocessor-trend-data`.

[106] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE T. on Acoustics, Speech, and Signal Processing*, 26(1), 1978. doi: 10.1109/TASSP.1978.1163055.

[107] Hiroaki Sakoe and Seibi Chiba. A similarity evaluation of speech patterns by dynamic programming. In *Nat. Meeting of Institute of Electronic Communications Engineers of Japan*, volume 136, 1970.

[108] Jorge Martínez Sánchez, Álvaro Váquez Álvarez, David López Vilariño, Francisco Fernández Rivera, José Carlos Cabaleiro, and Tomás Fernández Pena. Fast Ground Filtering of Airborne LiDAR Data Based on Iterative Scan-Line Spline Interpolation. *Remote Sensing*, 11:23, 2019. ISSN 2072-4292. doi: 10.3390/rs11192256.

[109] Steven C. Schachter. *Epilepsy in Our Experience: Accounts of Health Care Professionals*. Oxford University Press (OPS), 2007.

[110] Steven C. Schachter. *Epilepsy in Our Words: Personal Accounts of Living with Seizures*. Oxford University Press (OPS), 2007.

[111] Donald L. Schomer and Fernando H. Lopes da Silva. *Niedermeyer's Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Oxford University Press, 11 2017. ISBN 9780190228484. doi: 10.1093/med/9780190228484.001.0001. URL `https://doi.org/10.1093/med/9780190228484.001.0001`.

[112] Jie Shan and Charles K. Toth, editors. *Topographic Laser Ranging and Scanning: Principles and Processing*. CRC Press, 2 edition, 2018.

[113] Ali Shoeb and John Guttag. Application of machine learning to epileptic seizure detection. In *Int. Conf. on Machine Learning*, ICML'10, page 975–982, 2010. ISBN 9781605589077.

[114] Simon D. Shorvon and Torbjörn Tomson. Sudden unexpected death in epilepsy. *The Lancet*, 378:2028–2038, 2011.

[115] Mohammad Khubeb Siddiqui, Ruben Morales-Menendez, Xiaodi Huang, and Nasir Hussain. A review of epileptic seizure detection using machine learning classifiers. *Brain Informatics*, 7(1):5, 2020. doi: 10.1186/s40708-020-00105-1.

[116] SL Smith, DA Holland, and PA Longley. The importance of understanding error in lidar digital elevation models. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 35: 996–1001, 2004.

[117] D. Sopic, T. Teijeiro, D. Atienza, A. Aminifar, and P. Ryvlin. Personalized seizure signature: An interpretable approach to false alarm reduction for long-term epileptic seizure detection. *Epilepsia*, Feb 2022.

[118] Statista. Internet of Things (IoT) - Worldwide. `https://www.statista.com/outlook/tmo/internet-of-things/worldwide`, 2023. Accessed: 2024-02-27.

[119] Faisal Tariq, Muhammad R. A. Khandaker, Kai-Kit Wong, Muhammad A. Imran, Mehdi Bennis, and Merouane Debbah. A speculative study on 6g. *IEEE Wireless Communications*, 27(4):118–125, 2020. doi: 10.1109/MWC.001.1900488.

[120] TechHQ. China is planning its biggest state-backed chip fund yet. `https://techhq.com/2023/09/what-does-china-planning-its-biggest-state-backed-chip-fund-mean/`, September 2023. Accessed: 2024-02-27.

[121] The Khronos SYCL Working Group. SYCL 2020 specification (revision 8). `https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html`, 2023. Accessed: 2024-03-03.

[122] The Next Platform. Intel Downplays Hybrid CPU-GPU Compute Engines, Merges NNP into GPU. `https://www.nextplatform.com/2023/06/08/intel-downplays-hybrid-cpu-gpu-compute-engines-merges-nnp-into-gpu/`, June 2023. Accessed: 2024-02-27.

[123] The Royal Institute of British Architects. Town Planning Conference. `https://archive.org/details/transactions00town/page/372/mode/2up?q=%22geometric+widening%22`, 1910. Accessed: 2024-02-27, Page: 372.

[124] The White House. Fact sheet: The president's budget for fiscal year 2024. `https://www.whitehouse.gov/omb/briefing-room/2023/03/09/fact-sheet-the-presidents-budget-for-fiscal-year-2024/`, March 2023. Accessed: 2024-02-27.

[125] Maria Tito, Mercedes Cabrerizo, Melvin Ayala, Prasanna Jayakar, and Malek Adjouadi. A comparative study of intracranial eeg files using nonlinear classification methods. *Annals of Biomedical Engineering*, 38:187–199, 2009. URL `https://api.semanticscholar.org/CorpusID:228076`.

[126] Tohoku University Takizawa Lab. neoSYCL: A SYCL Implementation by Tohoku University Takizawa Lab. `https://github.com/Tohoku-University-Takizawa-Lab/neoSYCL`. Accessed: 2024-02-27.

[127] University of Heidelberg. AdaptiveCpp: A Project for Adaptive C++ Programming. `https://github.com/AdaptiveCpp/AdaptiveCpp`. Accessed: 2024-02-27.

[128] Jose Antonio Urigüen and Begoña Garcia-Zapirain. EEG artifact removal—state-of-the-art and guidelines. *J. of Neural Engineering*, 2015. doi: 10.1088/1741-2560/12/3/031001.

[129] van der Merwem Dirk and Johan Meyer. Towards automatic digital surface model generation using a graphics processing unit. In *AFRICON 2009*, pages 1–6, 2009. doi: 10.1109/AFRCON.2009.5308102.

[130] Emma Vargo, Le Ma, He Li, Qingteng Zhang, Junpyo Kwon, Katherine M. Evans, Xiaochen Tang, Victoria L. Tovmasyan, Jasmine Jan, Ana C. Arias, Hugo Destaillats, Ivan Kuzmenko, Jan Ilavsky, Wei-Ren Chen, William Heller, Robert O. Ritchie, Yi Liu, and Ting Xu. Functional composites by programming entropy-driven nanosheet growth. *Nature*, 623(7988):724–731, November 2023. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-023-06660-x. URL `https://www.nature.com/articles/s41586-023-06660-x`.

[131] Michael Voss, Rafael Asenjo, and James Reinders. *Pro TBB: C++ Parallel Programming with Threading Building Blocks*. Apress, 2019.

[132] Wikipedia. Accelerating change. `https://en.wikipedia.org/wiki/Accelerating_change`, 2024. Accessed: 2024-02-27.

[133] H. Wong, J. G. Koomey, S. Berard, and M. Sanchez. Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing*, 33(03):46–54, jul 2011. ISSN 1934-1547. doi: 10.1109/MAHC.2010.28.

[134] *Optimizing brain health across the life course*. World Health Organization, August 2022.

[135] Chen Xie, Lucas McCullum, Alistair Johnson, Tom Pollard, Brian Gow, and Benjamin Moody. Waveform database software package (wfdb) for python, Jan 2023. URL https://physionet.org/content/wfdb-python/4.1.0/.

[136] Xilinx. triSYCL: An Implementation of the SYCL Specification. https://github.com/triSYCL/triSYCL. Accessed: 2024-02-27.

[137] Chin-Chia Michael Yeh, Nickolas Kavantzas, and Eamonn Keogh. Matrix Profile IV: Using weakly labeled time series to predict outcomes. *Proc. VLDB Endow.*, 10(12):1802–1812, 2017. ISSN 2150-8097. doi: 10.14778/3137765.3137784.

[138] Morteza Zabihi et al. Patient-specific epileptic seizure detection in long-term EEG recording in paediatric patients with intractable seizures. In *IET Intelligent Signal Processing Conf. (ISP 2013)*, pages 1–7, 2013. doi: 10.1049/cp.2013.2060.

[139] I.C. Zibrandtsen, P. Kidmose, and T.W. Kjaer. Detection of generalized tonic-clonic seizures from ear-eeg based on emg analysis. *Seizure*, 59:54–59, 2018. ISSN 1059-1311. doi: https://doi.org/10.1016/j.seizure.2018.05.001. URL https://www.sciencedirect.com/science/article/pii/S1059131118300943.