National Technical University of Athens
School of Electrical & Computer Engineering
Division of Communication, Electronic and Information Engineering

Hardware Acceleration Techniques for Computation and Data Intensive Machine Learning and Bioinformatic Applications

Ph.D. Thesis

Konstantina I. Koliogeorgi

Athens, May 2023

....
Konstantina Koliogeorgi
Ph.D. Electrical & Computer Engineer N.T.U.A.

This Ph.D. thesis was cofinanced by the E.C. funded projects H2020 AEGLE and H2020 EVOLVE.

National Technical University of Athens
School of Electrical & Computer Engineering
Division of Communication, Electronic and
Information Engineering
Microprocessors and Digital Systems Lab

Hardware Acceleration Techniques for Computation and Data Intensive Machine Learning and Bioinformatic Applications

Ph.D. Thesis
of
Konstantina I. Koliogeorgi

Supervising Committee: Dimitrios Soudris
Kiamal Pekmestzi
Georgi Gaydadjiev

Approved by the advisory committee on May 2, 2023.

....	....	....
Dimitrios Soudris	Kiamal Pekmestzi	Georgi Gaydadjiev
Professor N.T.U.A	Professor N.T.U.A	Professor TU Delft

....	....
Sotirios Xydis	Dionisios Pnevmatikatos
Asst. Professor N.T.U.A	Professor N.T.U.A
....	....
Leonidas Alexopoulos	Onur Mutlu
Professor N.T.U.A	Professor E.T.H.Z

Athens, May 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

Τεχνικές Επιτάχυνσης σε Hardware για εφαρμογές Τεχνητής Νοημοσύνης και Βιοπληροφορικής απαιτητικές σε Υπολογισμούς και Δεδομένα

Διδακτορική Διατριβή
της
Κωνσταντίνας Ι. Κολιογεώργη

Αθήνα, Μάιος 2023

....
Κωνσταντίνα Ι. Κολιογεώργη
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και
Συστημάτων Πληροφορικής
Εργαστήριο Μικροϋπολογιστών

Τεχνικές Επιτάχυνσης σε Hardware για εφαρμογές Τεχνητής Νοημοσύνης και Βιοπληροφορικής απαιτητικές σε Υπολογισμούς και Δεδομένα

Διδακτορική Διατριβή
της
Κωνσταντίνας Ι. Κολιογεώργη

Συμβουλευτική Επιτροπή: Δημήτριος Σούντρης
Κιαμάλ Πεκμεστζή
Georgi Gaydadjiev

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 2^{η} Μαΐου 2022.

....	....	....
Δημήτριος Σούντρης	Κιαμάλ Πεκμεστζή	Georgi Gaydajiev
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Professor TU Delft

....	....
Σωτήριος Ξύδης	Διονύσιος Πνευματικάτος
Επ.Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.
....	....
Λεωνίδας Αλεξόπουλος	Onur Mutlu
Καθηγητής Ε.Μ.Π	Professor ETHZ

Αθήνα, Μάιος 2023

Abstract

In this thesis, we focus on the hardware acceleration of two representative applications of modern healthcare: a ML-based prediction analysis and Read Alignment of genomic data. Both fields experience an intense growth in the latest decades and generate an immense amount of raw data. Creating value and making decisions based on these data have proved to be a challenging task as both the datasets as well as the computational intensity of the algorithms continue to escalate. To cope with this issue, High Performance techniques such as hardware acceleration have been examined. There is a great surge of works that leverage different programming models and frameworks to develop efficient FPGA-based accelerators, thanks to the bit-level customization capabilities of the devices. However, the frameworks available for programming such devices cannot always straightforwardly fully exploit the acceleration prospects of the applications. Furthermore, in complex applications existing solutions are characterized by a narrow view on real integration aspects, such as system wide communication and accelerator call overheads. In the current research work, the core contribution is based on the delivery of efficient solutions through strategic exploration of the design space and the synergy of hardware and software code modifications. The first application that this thesis examines is efficient hardware acceleration of Support Vector Machine (SVM) classifiers. SVMs have played a crucial role in providing data fusion and high accuracy classification solutions for various, complex, non-linear problems. In this thesis, we explore an application that SVM hardware co-processors perform classification for ECG signal arrhythmia detection. The proposed methodology for accelerating the SVM has been implemented as a framework on top of the state-of-art Vivado High-Level Synthesis (HLS) tool. We propose a systematic two-level approach for SVM acceleration, which first optimizes the global structure of the original SVM's behavioral description to assist the tool in infering the inherent data- and instruction-level parallelism of the algorithm. The second level of optimization further refines the design through a targeted design exploration that matches the accelerator's memory architecture to its computation and memory access patterns. In the second part of the thesis, we study the effect of acceleration techniques on one of the major bottlenecks of a typical genomic pipeline, which is short read alignment. In our study we perform extensive profiling on a popular aligner and identify the bottleneck within alignment as the string-matching algorithm Smith-Waterman. Our approach is to provide a dataflow implementation for this task that targets FPGA devices by taking into account the implications of integrating the accelerator in the original software tool. We therefore present GANDAFL, a novel genome alignment dataflow architecture for Smith-Waterman Matrix-fill and Traceback stages to perform high throughput short-read alignment on Next Generation Sequencing data. We then propose a radical software restructuring to widely-used Bowtie2 aligner that implements an aggregation-batching strategy and feeds the accelerator in high-throughput streaming fashion with minimized transfer and call overheads. The standalone solution delivers up to $\times 116$ and $\times 2$ speedup over state-of-the-art software and hardware accelerators respectively and GANDAFL-enhanced Bowtie2 aligner delivers a $\times 1.9$ speedup. We also examine an alternative approach to accelerating short read alignment. We introduce a high throughput alignment system that combines Banded SmithWaterman accelerators and pre-filtering for alignment optimization by introducing a profile-driven accelerator methdology. Extensive profiling of genomic datasets reveals low edit thresholds that can be leveraged by a heuristic of SmithWaterman, i.e. Banded SmithWaterman, to create resource-efficient accelerators that are customized to the edit profile of the input. We therefore design and deliver a highly optimized dataflow implementation for Banded Smith-Waterman seed-extension targeting FPGA devices, which is leveraged within a multi-dataflow accelerated system. The multi-dataflow system covers the full range of edits and therefore achieves both high throughput as well as high accuracy alignments. The evaluation shows that the proposed Banded Smith-Waterman accelerator delivers a $\times 34$ speedup over state-of-the-art software aligners and $\times 1.53$ and $\times 3$ over state-of-the-art dataflow and RTL SmithWaterman accelerators respectively. The multi-dataflow system delivers average speedups of $\times 1.8$ over state-of-art multi-accelerator FPGA solutions that employ generic and input-agnostic accelerators.

Keywords: FPGA Acceleration, High Level Synthesis, Design Space Exploration, Code transformation, Machine Learning, Next-Generation Sequencing, Short Read Alignment, SW/HW Co-design, Dataflow computing

Περίληψη

Σε αυτή τη διατριβή επικεντρωνόμαστε στην υλοποίηση υλικού επιτάχυνσης για δύο αντιπροσωπευτικές εφαρμογές του σύγχρονου τομέα της υγείας: μια ανάλυση πρόβλεψης που βασίζεται στη μηχανική μάθηση και η ευθυγράμμιση ανάγνωσης γονιδιωματικών δεδομένων. Και οι δύο τομείς βιώνουν έντονη ανάπτυξη τις τελευταίες δεκαετίες και παράγουν έναν τεράστιο όγκο ακατέργαστων δεδομένων, πλούσιο σε πληροφορία. Η ερμηνεία και η λήψη αποφάσεων βασισμένων σε αυτά τα δεδομένα έχουν αποδειχθεί δύσκολες εργασίες καθώς τα δεδομένα και η υπολογιστική πολυπλοκότητα των αλγορίθμων αυξάνονται εκθετικά. Για να αντιμετωπιστεί αυτό το πρόβλημα, έχουν εξεταστεί τεχνικές υψηλής απόδοσης όπως η επιτάχυνση σε hardware. Υπάρχει μια πληθώρα ερευνητικών εργασιών που αξιοποιούν διαφορετικά μοντέλα προγραμματισμού για να αναπτύξουν αποτελεσματικούς επιταχυντές βασισμένους σε FPGA, χάρη στην ευελιξία προγραμματισμού τους σε επίπεδο βιτ. Ωστόσο, τα διαθέσιμα μοντέλα προγραμματισμού για την προγραμματισμό τέτοιων συσκευών δεν μπορούν πάντα να εκμεταλλευτούν πλήρως τις προοπτικές επιτάχυνσης των εφαρμογών με απλό τρόπο. Επιπλέον, σε πολύπλοκες εφαρμογές, οι υπάρχουσες λύσεις χαρακτηρίζονται από μια περιορισμένη οπτική στην ενσωμάτωση των επιταχυντών σε ένα ρεαλιστικό σύστημα, όπως η επικοινωνία σε επίπεδο συστήματος και οι πρόσθετοι χρόνοι κλήσης των επιταχυντών. Στο τρέχον διδακτορικό, η κύρια συνεισφορά βασίζεται στην παροχή αποτελεσματικών λύσεων μέσω της στρατηγικής εξερεύνησης του χώρου σχεδιασμού και της συνέργιας βελτιστοποιήσεων του κώδικα τόσο σε επίπεδο υλικού όσο και λογισμικού.

Η πρώτη εφαρμογή που εξετάζεται σε αυτή τη διατριβή είναι η αποδοτική επιτάχυνση υλικού των ταξινομητών Συππορτ ἕστορ Μαςηινε (SVM). Σε αυτήν τη διατριβή, εξετάζουμε μια εφαρμογή στην οποία οι επιταχυντές υλικού SVM εκτελούν ταξινόμηση για την ανίχνευση αρρυθμιών σήματος ECG. Η προτεινόμενη μεθοδολογία για την επιτάχυνση του SVM έχει υλοποιηθεί χρησιμοποιώντας το εργαλείο Vivado High-Level Synthesis (HLS). Προτείνουμε μια συστηματική προσέγγιση δύο επιπέδων για την επιτάχυνση του SVM, η οποία πρώτα βελτιστοποιεί τη γενική δομή της αρχικής περιγραφής συμπεριφοράς του SVM για να βοηθήσει το εργαλείο να αναγνωρίσει τον εγγενμή παραλληλισμό σε επίπεδο δεδομένων και εντολών του αλγορίθμου. Το δεύτερο επίπεδο βελτιστοποίησης βελτιώνει επιπρόσθετα το σχεδιασμό μέσω μιας στρατηγικής εξερεύνησης του χώρου σχεδιασμού που σχεδιάζει τη μνήμη του επιταχυντή βάσει των μοτίβων υπολογισμού και πρόσβασης στη μνήμη του.

Στο δεύτερο μέρος της διπλωματικής εργασίας, μελετάμε την επίδραση των τεχνικών επιτάχυνσης σε ένα από τα πιο υπολογιστικά απαιτητικά κομμάτια της επεξεργασίας γονιδιώματος, που είναι η ευθυγράμμιση ακολουθιών ΔΝΑ στο ανθρώπινο γονιδίωμα. Εκτελούμε ανάλυση της απόδοσης ενός εργαλείου αλληλούχισης (το Bowtie2) και εντοπίζουμε τον αλγόριθμο Smith-Waterman ως το πιο χρονοβόρο κομμάτι. Η προσέγγισή μας είναι να παρέχουμε μια υλοποίηση ροής δεδομένων που στοχεύει συσκευές FPGA λαμβάνοντας υπόψη τις συνέπειες της ενσωμάτωσης του επιταχυντή στο εργαλείο αλληλούχισης και επομένως σε ένα πραγματικό σύστημα. Προτείνουμε το GANDAFL, μια νέα αρχιτεκτονική ροής δεδομένων ευθυγράμμισης γονιδιώματος για τον Smith-Waterman για την εκτέλεση ευθυγράμμισης υψηλής απόδοσης σε δεδομένα αλληλουχίας επόμενης γενιάς. Στη συνέχεια, προτείνουμε μια ριζική αναδιάρθρωση του κώδικα του Bowtie2 η οποία ομαδοποιεί πολλά μεμονωμένα αιτήματα αλληλούχισης και τα τροφοδοτεί στον επιταχυντή με υψηλής ρυθμό απόδοσης ελαχιστοποιώντας έξοδα μεταφοράς και κλήσεων. Ο επιταχυντής προσφέρει έως και 116 και 2 φορές επιτάχυνση αντίστοιχα σε σύγκριση με πρόσφατους επιταχυντές λογισμικού και υλικού, αντίστοιχα, και η βελτιωμένη με GANDAFL ευθυγράμμιση Bowtie2 προσφέρει επιτάχυνση 1,9 επί του συνολικού συστήματος. Τέλος εξετάζουμε μια εναλλακτική προσέγγιση, η οποία συνδυάζει μια ευριστική υλοποίηση του Smith-Waterman και ένα στάδιο φιλτραρίσματος των αρχικών δεδομένων. Μελέτη των δεδομένων εισόδου υποδεικνύει ότι η αλληλούχιση συνήθως είναι ακριβής και εντοπίζεται μικρός αριθμός διαφοροποιήσεων από το ανθρώπινο γονιδίωμα. Αυτό μειώνει το χώρο αναζήτησης των λύσεων και μας επιτρέπει να χρησιμοποιήσουμε τον ευριστικό Banded Smith Waterman ο οποίος επιτελεί την ίδια λειτουργία, εντοπίζει λιγότερες διαφοροποιήσεις και καταναλώνει λιγότερους πόρους στο υλικό. Προτείνουμε λοιπόν ένα σύστημα που πλέον αποτελείται από πολλούς επιταχυντές και καλύπτει έως έναν αριθμό διαφοροποιήσεων ενώ εντοπίζει πλέον τις αλληλουχίσεις με ταχύτερο ρυθμό. Το προτεινόμενο σύστημα αποδίδει επιτάχυνση έως 34 φορές σε σχέση με λογισμικά ενώ είναι έως 3 φορές ταχύτερο από σύγχρονους επιταχυντές.

Λέξεις Κλειδιά: Επιτάχυνση Υλικού, Γλώσσες Σύνθεσης Υψηλού Επιπέδου, Μετασχηματισμός Κώδικα, Διερεύνηση Χώρου Σχεδίασης, Μηχανική μάθηση, Αλληλούχιση γονιδιώματος, Next-Generation Sequencing, Short Read Alignment, SW/HW Co-design, Dataflow computing

Acknowledgements

This section of my thesis is dedicated to expressing my gratitude to all people that have contributed to my efforts and completion of my Ph.D. studies. It is very challenging to capture the thoughts and feelings that occur upon reflecting over the past years. I can only hope that the next words, despite their simplicity and lack of originality, convey the sincerity and depth of my gratitude.

To begin with, I would like to express my graditude for being a member of the Microprocessors and Digital Systems Lab (MicroLab) of N.T.U.A.. First and foremost, I would like to thank my advisor Professor Dimitrios Soudris for trusting me and offering me the opportunity to pursue my Ph.D. degree in such an inspiring and productive environment as is MicroLab. His constant technical and moral support and genuine interest has encouraged me and guided me through all steps of my studies. The lessons learned during this mentorship will stay close to me and continue to help me cope with the new challenges in my life. Next I would like to specially acknowledge the contribution of Sotirios Xydis, with whom I closely collaborated since my diploma thesis. His insightful remarks on my research and our intriguing discussions were inspiring and crucial for the completion of my thesis. Hopefully his love for research and work ethics are reflected on my work from now on.

My Ph.D. studies would not be the same without the Ph.D. candidates, researchers and administrative staff that comprise MicroLab. I would like to thank all of them for being part of one of the most challenging accomplishments of my life and some of my fondest memories. I consider them my friends and they have trully made MicroLab feel like a second home. With that said, I would like to specially thank Dimosthenis Masouros, Georgios Zervakis, Giannis Stratakos, Haris Marantos, Vasilis Leon, Manolis Katsaragakis, Giannis Oroutzoglou, Achilleas Tzenetopoulos, Aggelos Ferikoglou, Christos Lamprakos, George Armeniakos, George Lentaris and Zefi Skini.

I could not omit to thank all of my friends that stood by me throughout these years. They have patiently supported me through hard work and deadlines, encouraged me to reach my goals and filled my life outside Microlab with beautiful moments. I am sorry for missing out on some of them and grateful for your understanding and love. A special thanks to Anastasia, Katerina, Christos, Konstantina M, Konstantina S, Sotiria Fytraki, Sotiria Filandra, Rania and Rafailia.

Finally, I would like to thank my parents, Ioannis and Matoula, and my sister, Roza, for their unconditional love and support. They are the ones that were closest to me during this venture, offering me their support and never losing faith in me. Without them, I would not have the perseverance to fulfil my goals.

Μαμά, Μπαμπά και Ρόζα,

Σας ευχαριστώ για όλα. Ελπίζω να σας δίνω χαρές και να στέκομαι δίπλα σας όπως εσείς σε εμένα.

Κωνσταντίνα

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

1.1. Big Data Overview and Challenges

Technological advancements and novelties have led to an exponential growth of data availability and have signaled the start of the Information or Digital age[1]. As technology evolved, the aggregated data initially originated from newspapers, radio and television and in later years by computers, Internet and mobile phones [3–5]. What has dramatically accelerated the establishment of this new era is the speed of data transmission and ease of accessibility by all humans thanks to new technologies.

As the capacity of computer systems and devices to generate data continued to grow, the term of *Big Data* was gradually adopted. It is not possible to define Big Data by the size of data it refers to, as this is relative to the storage and processing power of computer systems. As the data production and availability grows exponentially, computing systems evolve and adapt to accommodate the data curation and processing requirements. Therefore, what was once considered to be Big data, could not possibly fall into that category nowadays, that we have entered the so-called Zettabyte Era. A zettabyte is a measure of storage capacity, which equals 1000^7 (1,000,000,000,000,000,000,000 bytes), which is equal to a thousand exabytes or a trillion gigabytes. Fig.1.1 illustrates the exponential data growth starting from 2010 and including estimates until 2025. In 1999, the total size of available data was 1.5 exabytes and reached 160 exabytes within seven years. Therefore, between the late 1990s and early 2000s, a dataset of size 1GB would qualify as big data. Total data storage capacity in 2019 reached almost 50 zettabytes and it is estimated to come close to 200 zettabytes by 2025 [2]. Interestingly, 90% of the data has been created only since 2016. The size of big data is therefore so fluid and quickly redefined, that a possible definition could be that of information that's so extensive or complex that it's difficult or impossible to process using traditional methods and technology [3].

[1]Information Age
[2]How Big is Big Data
[3]What is Big Data

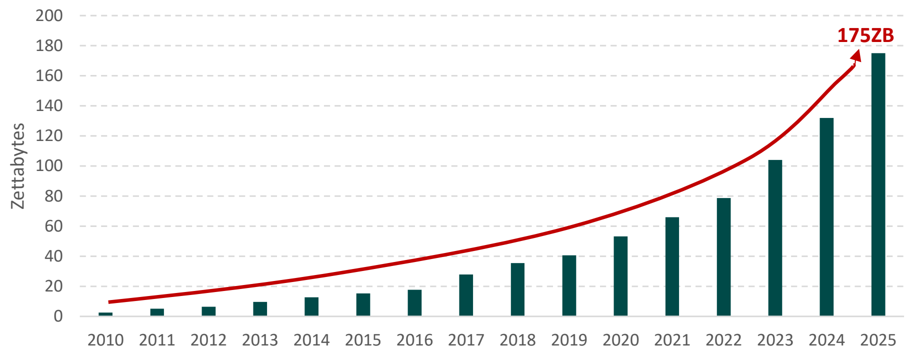


Figure 1.1.: Annual Size of the Global Datasphere. Source: Data Age 2025, sponsored by Seagate withdata from IDC Global Datasphere.

Big Data Domains: The onslaught of information and data has been so fierce that it has managed to define modern societies and cause a major shift in the dynamics of global economy, as an increasing number of businesses has to handle big data or has been created with this purpose. Companies with activity in this market typically need some kind of software, i.e. Big Data analytics. Depending on the customer, these analytics could be a prediction model, risk analysis or visualization components. Apart from software, users also need to acquire special hardware and equipment to efficiently participate in the Big Data analysis chain such as connected devices, network equipment, mobile devices. Such matters demonstrate such complexity that led to the establishment of consultation companies that assist others in managing big data, building a big data infrastructure e.t.c.. These activities add up to a very prosperous market that was worth 206.95 USD billion in 2020. As the big data growth trend does not show any signs of slowing down, there are projections that estimate the market growth to reach 549.73 billion USD by 2028[4].

Big Data are generated from various aspects of human activities. Fields like education, banking, retail, agriculture, healthcare, IT and telecommunications have been generating data exponentially and therefore require big data solutions for efficient processing. For example in agriculture, data from sensors, GPS-equipped tractors, satellites, soil sensors can be leveraged to perform risk assessments, crops optimization and prediction e.t.c.. In the social sciences, there is an enormous volume of data to evaluate coming from social media platforms (Facebook, Instagram), chat applications (WhatsApp), and services such as Youtube. The BFSI (Banking, Financial Services and Insurance) domain has leveraged big data analytics to improve the quality of customer service[5] (e.g. tracking their activity to provide for resources when required) as well as the decision-making system, risk management processes, retention strategies[6] in order to maximise gains. In the telecommunication and media domain, big data analytics are leveraged to optimize the network,

[4]Big Data Analytics Market Report, 2021-2028, Fortune Business Insight.
[5]Big data as a tool to improve customer experience.
[6]Your Go-to Guide to Big Data Analytics in Banking.

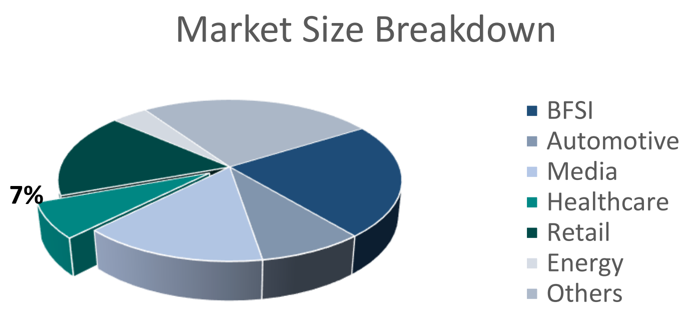


Figure 1.2.: Global Big data analytics market share distribution,2020. Source: Fortune Business Insights.

perform preventive diagnostics, prevent fraud[7] e.t.c..

Big Data in Healthcare: Fig.1.2 represents a distribution percentage for the fields that hold the larger shares in the market, as those were estimated in 2020. Healthcare holds a considerable portion of the market share as big data analytics are required to cope with the recent fast data growth. Until recently, all form of medical data (such as patient history, clinical data, results of exams) were stored in a paper file system. The digitization of all clinical exams and medical records and the advent of new technologies has led to a data abundance and has also created great potential for improved healthcare services and flourishing of research activities.

Fig.1.3 depicts the major types of data in the healthcare big-data repository as well as the main type of analyses that rely on these data. The Public Health Records and Electronic Health Records (EHRs) are an electronic version of the medical history of a patient and is available to public health providers in order to optimize the procedure of diagnosis and therapy of a patient. It includes demographic data, past diagnoses, medicines, allergies, immunizations and treatment plans [6]. Clinical data refer to both administrative data as well as data created during clinical practice. Data generated during clinical practice are mainly signal-based signals, i.e. time series, that are essential for monitoring e.g. ECG,EEG,ventilator signals, ICU data etc. A large part of these data also come from IoT devices, i.e. mainly health-tracking wearable devices, biosensors, clinical devices for monitoring vital signs, and others types of devices or clinical instruments. Clinical data also include images such as X-ray and CT scans, MRI, fMRI. Both signal-based and imaging data can assist in monitoring, warning of critical situations, diagnosis, drug development e.t.c. They are utilized both for diagnosis as well as health surveys and clinical trials.

Apart from data generated and utilized in the clinical practice, there is a huge part of health data also come from the study of diseases and human on a biological level. Genes, transcripts, proteins, metabolites, and other macro/ micro molecules systematically col-

[7]how telecom companies use big data analytics-Top 10 use cases.

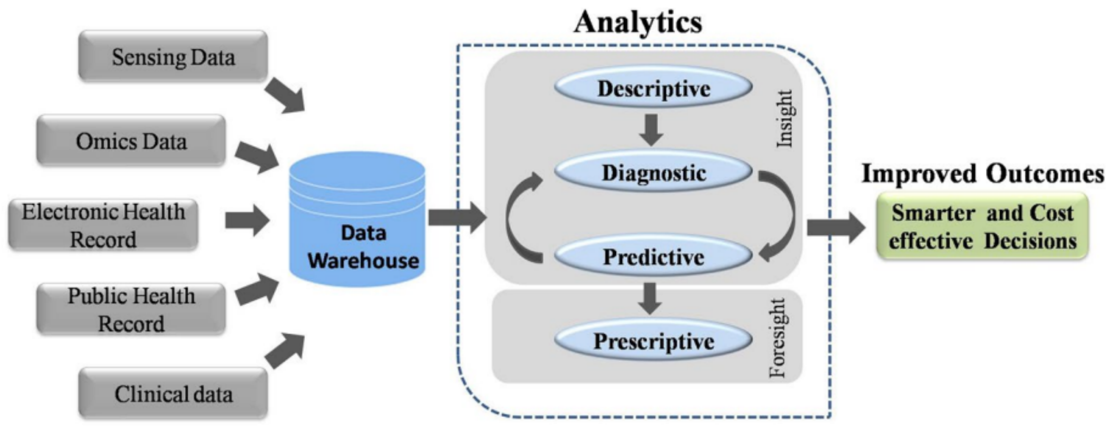


Figure 1.3.: Storage of massive amount of data from various sources and Big data Analytics in the Healthcare domain [1].

laborate to perform complex cellular processes. The advent of whole-genome sequencing and other high-throughput experimental technologies have created the data rich disciplines of genomics, transcriptomics, epigenomics, proteomics, metabolomics, phenomics e.t.c. These immense data sets make up the *omics* data [7]. Multi-omics data generated for the same samples can be combined to provide useful insights into the flow of biological information at multiple levels and thus can help in unraveling the mechanisms underlying the biological condition of interest. Therefore the study of omics straightforwardly affects biological research on a cellular level. However, the insights have had a impact on clinical applications too and have paved the way to drug-development applications and, ultimately, into personalized genomic medicine [8–11].

Healthcare big data analytics are developed to extract value out of this immense and diverse amount of data [12]. These analytics require experts that come from interdisciplinary fields such as biology, information technology, statistics and mathematics and work in synergy to provide meaningful analytics and improve healthcare services and treatments. There is a wide range of analytics developed with the intent to assist in the clinical practice as well as in the diagnosis and formation of the therapeutic schema. For example, **descriptive** analytics are leveraged to present data in an understandable manner and make it easy to detect patterns. Applications such as medical imaging are an integral part of medical practice and diagnosis. **Predictive** analytics have also been developed to guide the medical staff in making a diagnosis and choosing the most effective treatments based on previous experience and results. This can lead to accurate diagnosis and limit the number of redundant and expensive clinical exams. Therefore, the integration of big data in the healthcare shows promise for improving health outcomes while preserving a low cost per medical case.

Computational Challenges and Solutions: Handling and storing such an enormous size of big data poses a challenge in healthcare analytics, as does in the case of big data analytics in general. In terms of computational performance, the enormous datasets and the computational intensity of these analytics stress the limits of modern computing power. As data continue to grow exponentially, systems and algorithms should be constantly updated and improved to provide the computational power to solve big data problems.

An initial approach to big data challenges was to create Big data infrastructures that are essential for meeting the storage and computational requirements. Until recently, big data processing challenges were solved by ecosystems such as Hadoop [13–16] and Spark [17, 18]. Applications were also altered to run within MapReduce [19–22], which is a parallel programming model was developed to run applications with high scalability and fault tolerance on top of HDFS systems. To mitigate however learning curve and tedious problem-solving related with these technologies, most enterprises have migrated big data to the cloud. Cloud computing offers virtualized storage technologies and provides services with high reliability, scalability and autonomy. Several cloud environments have been set up to support the execution of big data analytics in healthcare e.t.c [23–25]. AWS, Microsoft Azure, and Google Cloud Platform offer pay-as-you-go services and allow companies to run data-intensive operations without the installation overhead and with unlimited scalability. A striking example is the deployment of powerful DRAGEN-IT platform on AWS and Azure to analyze next-generation sequencing data [26]. This trend is expected to continue and further develop into hybrid environments (i.e. both local premises and cloud access) and multi-cloud environments [8].

A more recent approach involves adoption of techniques from the High Performance Computing world. HPC systems have been dealing with complex data and compute intensive applications. The size of the data that HPC applications handle though follow the trend of the Big Data era. This leads to a convergence between the HPC and Big Data ecosystems, creating High Performance Data Analytics (HPDA) and platforms suitable for their execution [27–30]. In this convergence HPC brings approaches such as Message Passing Interface (MPI) [31] and OpenMP [32], possibly combined with accelerators, such as Graphics Processing Units (GPUs) or Field Programmable Gate Arrays (FPGAs). For example, several works have leveraged such programming models [33,34] and co-processors [35–37] to deliver accelerated tools in the genomics domain.

Big data analyitcs in healthcare face the additional challenge of handling heterogeneous and unstructured data as well as operating in a clinical setting that calls for quick and efficient decisions. In this context, machine learning has been extensively leveraged to cope with the unstructured raw data utilized and assist in finding patterns within them. Many decision making analyses rely on ML and AI to detect complex relationships and assist in the diagnosis procedure. The most common use of ML in medicine is in predictive analytics to detect potential abnormalities and assist physicians in making better clinical decisions. The extensive use of ML and AI in the medical domain is reviewed and

[8]The future of big data: 5 predictions from experts for 2020-2025

motivated in several works [38–41].

In the next paragraphs, we focus on two important categories of healthcare analytics that cope with an enormous set of data and face performance bottlenecks, i.e. ML-based predictive analytics and genomics. We first elaborate on the importance and impact of Machine Learning and Genomics in healthcare and motivate the need for employing optimization techniques from the HPC world to alleviate the introduced challenges in two separate use cases.

1.2. Machine Learning and Bioinformatics in Healthcare

Machine Learning in HealthCare: Challenges and Trends

The inherent purpose and characteristic of machine learning is the ability to process data in order to discover underlying patterns, relationships between different entities as well as insights within human activities and behavior. As such, machine learning has an essential role in the big data processing chain and analytics [42]. In fact, the data deluge of the last decades as well as the variety and complexity of big data has spurred on an impressive growth in the machine learning domain. Since healthcare has proved to be one of the most flourishing big data domains, machine learning has been established as an essential tool in health data manipulation and analysis. Machine learning solutions fit both the big data nature of healthcare data as well as the need of healthcare to find patterns and correlations within unstructured raw medical data.

The size of the datasets in combination with the computational complexity of uncovering relations within such volume has truly challenged the capabilities of machine learning techniques and has led to the development of new approaches and algorithms [43] in order to meet the requirements and successfully extract patterns and build predictive models. Databases, data mining, information retrieval, machine learning, deep learning, AI are only a few of the fields that created opportunity for ML to flourish[9]. The use of AI and ML in healthcare however also encounters challenges specific to the field and use cases [38]. From a technical perspective, it is challenging to integrate data from different sources (i.e. healthcare units) and account for the heterogeneity and potential bias/noise in each one. Similarly, it is not feasible to require a golden standard as it should be acquired from the general population and reviewed by medical practitioners. Problems also arise when integrating machine learning models in a medical environment, as the results cannot be translated effectively in a way understandable by health staff. Developing a big data infrastructure or cloud services for sharing data among different healthcare centers is still challenging as they need to adhere to a universal standard

[9]When Machine Learning meets Big Data

representation of data and privacy regulations. Lastly, the adoption of proposed ML solutions requires validation and improvement through clinical trials and studies to lead to successful adoption in the medical world.

Despite the difficulties, there are on-going and promising works that leverage ML for solving healthcare problems. One of the primary areas of applying machine learning in healthcare is *information extraction.* AI programs and tools such as NLP [44] or character recognition were leveraged from the start to extract information from free text or speech and assist in digitization of medical data. Machine learning has been extensively leveraged for *prediction* tasks in application where real-time processing of signals and decision-making are critical. A representative example is the framework described in [45], which trains an ML model with time-series signals (e.g. heart rate, systolic BP,respiratory rate etc) to provide a real-time early warning for septic shock. Similarly, the authors in [46] develop a real-time system for timely detection of heart disease based on streaming signals originating from IoT and wearable devices. ML has also been established for detection or prediction tasks where real-time processing is not essential. For example, it is leveraged in image analytics to detect abnormalities in high resolution medical images (e.g. CT, MRI, fMRI, EEG) [47]. Therefore, it can prove very useful to the diagnosis process even at a consultation level or as a measure of precaution to avoid misdiagnosis [1].

Overall, the importance of incorporating ML analytics in healthcare is highlighted by the magnitude and critical nature of diagnostic and prognostic applications. Epileptic seizure prediction [48], breast cancer diagnosis [49, 50], heart disease diagnosis [51], research for brain diseases [52] are only a few of the prominent use cases that leverage ML and AI to improve the quality of healthcare services. However, due to the big data nature of medical records and signals, the need to identify complex patterns within the data and often the real-time requirements, ML-healthcare analytics have intense memory and computational requirements. As a result, there is rich literature that studies machine learning techniques applied in healthcare from the scope of optimization and performance enhancement.

In general, machine learning methods used in healthcare analytics share the same optimization techniques with ML solutions for big data analytics. A common practice to achieve scalability and data parallelism [43], is to employ big data frameworks such as Hadoop [53] and Spark [54]. Another approach requires implementation of ML models to run through the MapReduce programming model [55–60]. Cloud computing is also extensively used to deliver efficient healthcare services [61, 62]. Apart from specialized big data techniques and cloud computing, optimization solutions also leverage the parallelism and scalability of High Performance Computing and specifically of powerful hardware resources. Hardware optimizations are examined on architecture level, mixed-signal circuits and advanced technologies such as efficient memories e.t.c. [63]. As far as platforms are concerned, GPUs, FPGAs and ASICs devices are targeted by several works [64]. Increased interest has been also attracted by hybrid solutions that incorporate GPU or FPGA accelerators for ML models in data centers and at the edge [65, 66].

Bioinformatics: Challenges and Trends

Bioinformatics is an interdisciplinary field focusing on the application of statistical and computational methodologies to manage and interpret biological data [67]. An essential part of bioinformatics is the study of DNA sequences for research or therapeutical purposes. Recent technological advances in high throughput technologies has led to an unprecedented wealth of genomic sequence data. The low cost of data generation has created an enormous amount of data, signaling the establishment of genomic data as a new unique type of big data [68].

This new era of genomic data deluge commenced with the completion of the Human Genome project, which for the first time generated an accurate sequence of 3 billion DNA bases covering 99% of the gene-containing regions[10]. Since then, many projects have been conducted to further understand areas of interest within the human genome e.g. the Cancer Genome Atlas and the Encyclopedia of DNA Elements which identify somatic mutations, mRNA expressions, histology slides for approximately 7000 human tumors[11] [68]. The enormous size of genomic and transcriptomic data is obvious, if we take into account that sequencing a single whole genome generates more than 100 gigabytes of data. Specifically, depending on the read length and coverage (i.e. the average number of times each base is read), when sampling a human individual the original file of reads in FASTQ is approximately 250 GB, the BAM file can be approximately 100 GB, the VCF file can be about 1 GB, and the annotated files can be approximately 1 GB as well [69]. The size escalates when millions of individuals are sequenced. The advent of mass spectrometry and nuclear magnetic resonance (NMR) spectroscopy has come to add another source of big data, by generating proteomic and metabolomic data, and establishing the term of *omics* data.

Omics studies have since established a significant role in the healthcare domain in the field of understanding, prevention as well as treatment of diseases. In fact, the integration and correlation among genomics, proteomics, metabolomics e.t.c., i.e. multi-omics analysis, paves the way for personalized medicine [70]. Genomic workflows are at the heart of omics analysis and involve a wide range of applications. Whole-genome sequencing (WGS) [71] is the basis of all genomic analysis and allows detection of common as well as rare genetic variants across the genome. Joint variant detection and genotyping [72] are developed to study genetic variants across a large population or even a wider society. Functional interpretation [73] of the molecular effects of genetic variants is also necessary in order to infer if a variant is expressed in the phenotype and responsible for a specific disease [74].

[10]Human Genome Project Results
[11]National Cancer Institute

An outstanding example of how genomics has impacted healthcare, is the integration of genomic data in cancer studies in order to understand the biological dynamics of cancers and identify risk factors, as well as predict therapeutic outcomes and prognosis [75].

The impact of shift towards personalized medicines and treatments has naturally led to a wider adoption of genomics technology. The capabilities of genomics have been further underlined by the COVID-19 outburst as the genome sequence of the virus is critical for developing effective vaccines and treatments[12]. This has strengthened the lead players in the market of genomics such as Illumina Inc.,Thermo Fisher Scientific Inc., Agilent Technologies[13] e.t.c. but it has also increased government funded projects and the establishment of many start-up businesses. This is reflected in Fig.1.4 in the trend for the genomics market in North America, which is expected to hold the highest market share. A similar trend is expected in the global market, whose size will grow from 23.11 billion USD in 2020 to 94.65 billion USD in 2028. A more modest estimate reports an expected market size of 84.57 billion USD by 2031[14].

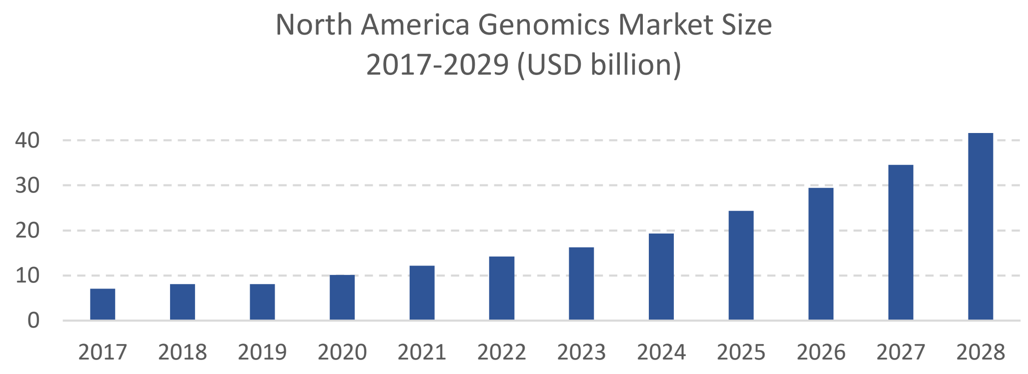


Figure 1.4.: Estimation for Genomics Market Size in upcoming years.

Challenges: This economic growth and need for new services requires the development of novel and efficient analytics to extract useful information from genomic data both for the research and industry domains. However, attributes of genomic data and workflows introduce several challenges that need to be overcome to efficiently process and analyse large genomic data sets [76, 77]. Firstly, the dramatic data growth calls for enormous databases to store data and share them across the research community. The challenge however does not solely lie on the data volume but also on the heterogeneity of data and lack of structure. An inherent trait of most genomic analyses is the need to discover patterns and an underlying structure within the raw data. This leads to sparse data structures, lack of data locality, irregular memory accesses and asynchronous updates to shared structures [78].

[12]Genomics Market Research Report 2021-2028, Fortune Business Insights.
[13]Grand View Research Genomics Market Size.
[14]Bloomberg Global Genomics Industry Update.

Trends: These challenges have yet to intensify as the sequencing cost further decreases and more complex analyses are required. The challenges can only be surpassed by leveraging high-end computing solutions and large-scale computational platforms to meet the memory and computational requirements of most genomic analyses. Several approaches have been pursued and different technologies leveraged to achieve quick and accurate results from genomic analytics. Mitigating bioinformatics analytics to Big Data infrastructures can greatly benefit the analysis of genomic data. Firstly, it is a very convenient solution as it can aggregate data from various different sources and store the data in virtualized storage technologies. A cloud computing system is also convenient and flexible as it can support different software libraries, development environments and visualization tools available as pre-installed software tools or containerized services. Cloud computing systems are also suitable candidates for genome analysis thanks to their distribution and scalability capabilities. A typical example of a distributed framework utilized in genomics analytics in the cloud is Hadoop [79] (HDFS - Hadoop Distributed Filesystem) , which separates the data into small fragments, distributes them across many cluster nodes, performs the computation on each node so that they are processed in parallel, and aggregates the results. The parallel processing of many small pieces of data is secured by MapReduce [80] programming model. A typical example of genomic analysis that leverages Hadoop is drug development using genomic and proteomic data [1]. Several other works deploy genomic frameworks on cloud infrastuctures as reviewed in the literature [81–85]. A recent example is *Genesis* framework, that provides an interface for performing SQL-like queries for data manipulation and an integrated hardware library of various genomic analysis stages [86]. Last but not least, cloud computing is also a cost efficient solution. In fact, cloud providers offer a pay-as-you-go policy that allows users to benefit from distributed computing and parallel programming without bearing the cost of building their own infrastructure. However, even these platforms have some adverse impacts, i.e. multi-core solutions increase the energy consumption whereas cloud solutions raise data privacy and ethics issues [87].

Despite the lack of regular accesses and data locality across the span of genomic analyses, each stage of a genomic pipeline usually leverages certain types of algorithms and therefore optimization HPC techniques for efficient execution. For the indexing stage, typical approaches are optimizing the data structure for storing the genome index as well as the search algorithm for locating seeds through this index, e.g. FM-index [88,89]. Hash-tables are also utilized to accelerate search operations in genome assembly and k-mer counting as well as graph theory techniques for efficient graph traversal within genome assembly [78]. The pre-alignment filtering stage is usually accelerated by proposing efficient algorithms that come from the HPC world and can deliver a quick and accurate prediction for redundant alignments. Such examples are the pigeonhole principle [90], q-gram filtering [91] and sparse dynamic programming [92].

The alignment stage holds the primary focus of most accelerating efforts targeting genomic pipelines as it usually forms a major bottleck. Extensive studies review the trends in proposed optimized systems such as specialized architectures, hardware accelerators and heuristic approaches [93]. Hardware-accelerated solutions specifically is one of the most popular HPC optimization for genomics. A plethora of works exist targeting different architectures from heterogeneous devices, e.g. FPGAs, to many-core accelerators such as GPUs and the Intel Xeon Phi. These devices are usually leveraged as a co-processor to off-load bottlenecks, but can also be utilized for end-to-end accelerated solutions that perform the same functionality as software genomic tools. Lately, in-memory computing has also been proposed for optimizing all genomic stages including the alignment [94–96].

1.3. Thesis Scope and Organization

The thesis scope is fully aligned with the trends described in Section 1.2 and aims to enrich the state of the art works that develop powerful co-processors. The goal is to examine offloading the bottlenecks to reprogrammable devices, i.e. FPGAs, through the meticulous design of efficient accelerators that exploit the existing parallelism and optimize communication with the overall system. The resulting accelerators can then be leveraged on local environments or can be deployed on large-scale cloud infrastructures to explore the results of the synergy of hardware acceleration and cloud computing technologies.

In this thesis, we focus on the hardware acceleration of two representative applications of modern healthcare: a ML-based prediction analysis using ECG signals and Read Alignment of genomic data. The basic HPC tool we utilize to deliver efficient solutions is *High Level Synthesis* programming targeting FPGA devices. With the proposed implementations, we aim to build upon existing solutions and suggest potential improvements.

We explore different aspects of optimization techniques regarding a single accelerator and leverage the inherent tools' capabilities to increase both instruction level and data level parallelism. In this context, we look for an efficient exploration of built-in optimization directives and strategy for applying them on the accelerator. However we do not limit our search by the tools straightforward solutions but rather find its inefficiencies and guide it in exploiting parallelism through manual source modifications. For complex applications, we do not restrict our efforts on a single module microarchitecture while ignoring the interaction with its environment. On the contrary, we adopt an holistic approach and view the design as part of a greater system. We prove that being mindful of integration implications is necessary for efficient architectural decisions as part of software-hardware co-design process. We propose techniques that are in tune with these objectives and apply them selectively to two major use cases, one from each domain. We also leverage more than one HLS frameworks and programming models in order to fully evaluate the capabilities of the available tools and demonstrate that the architectural decisions, instruction level parallelism and memory configuration schemas we apply can be agnostic of the selected vendor and software by a large part.

Based on these principles, we develop three accelerators using High Level Synthesis for the Machine Learning and Genomics domains. The thesis unfolds in two major directions:

An Exploration Framework for Efficient High-Level Synthesis of Support Vector Machines:Case Study on ECG Arrhythmia Detection for Xilinx Zynq SoC. A methodology for accelerating an SVM classifier has been implemented as a framework on top of the state-of-art Vivado High-Level Synthesis (HLS) tool. Our proposed SVM accelerator and methodology is validated for a healthcare usecase that performs classification for arrhythmia detection in ECG signals. We propose a systematic two-level approach that first optimizes the global structure of the original SVM's behavioral description to exploit the data- and instruction-level parallelism and then further refine it through a targeted design exploration of the tool's automatic optimization technique.

Acceleration of Short Read Alignment. The thesis presents two different approaches on accelerating the short read alignment problem. First, we present GANDAFL, a novel genome alignment dataflow architecture for Smith-Waterman Matrix-fill and Traceback stages to perform high throughput short-read alignment on NGS data. The implementation of Traceback on hardware is combined with a radical software restructuring of Bowtie2 aligner, that implements a batching aggregation strategy, diminishes data transfer and accelerator call overheads and allows for efficient integration of the accelerator within the aligner to deliver an end-to-end speedup. We then introduce a high throughput alignment system that combines Banded SmithWaterman accelerators and pre-filtering for alignment optimization by introducing a profile-driven accelerator methdology. The proposed system includes multiple accelerators with edit thresholds indicated by the input edit threshold distribution and leverages pre-filtering to guide candidate alignments to the appropriate accelerator. The adoption of these profile-driven upper limits and resource-efficient Banded SmithWaterman accelerators enables the provision of a highly parallel Banded Smith-Waterman system that accelerates read alignment while preserving the accuracy.

The rest of this thesis is organized as follows:

- Chapter 2 presents prior art of the examined fields and highlights the contributions of the Thesis.

- Chapter 3 provides a theoritical background on the domains discussed throughout the thesis. It presents the fundamental concepts and principles of the applications and introduces the basic capabilities of the utilized tools and frameworks.

- Chapter 4 presents the work conducted to accelerate the SVM for arrythmia detection based on an optimization strategy that leverages both source code optimizations and built-in tuning knobs of Vivado HLS tool.
- Chapter 5 presents a novel genome alignment architecture for Smith Waterman algorithm and the integration with popular Bowtie2 aligner.
- Chapter 6 presents a high throughput alignment system that is based on a profile-driven methodology and leverages multiple Banded SmithWaterman accelerators of different edit thresholds to deliver speedup while preserving alignment accuracy.
- Chapter 7 concludes this thesis by summarizing the presented results and discusses the future extensions of this work.

Chapter 2.

Thesis Contribution

In this thesis, we focus on two different healthcare applications and employ hardware acceleration techniques from the HPC domain for optimization purposes. Fig.2.1 presents a non-comprehensive taxonomy of applications in the convergence of the HPC and healthcare domain from the thesis perspective. We examine two major categories of healthcare data modalities and the respective analyses required to extract biological and clinical insights.

Omics data are generated by raw DNA samples or molecules through technologies such as sequencing and mass spectrometry. Each omics data, i.e. genomics, transcriptomics, proteomics, metabolomics etc, provide necessary and valuable information on mechanisms on multiple molecular levels. A hierarchy of data processing and analyses is required to extract this information from the initial samples. Multi-omics analysis is a popular bioinformatics approach that combines information from cellular up to metabolic level in order to understand the underlying mechanisms and leverage this knowledge for research and clinical purposes. This thesis scopes lies within the secondary analysis of genomic data, which performs alignment of DNA fragments in order to provide the full sequence for a sample and subsequently determine genetic variants.

Data generated from medical equipment during everyday clinical practice are another major category of healthcare data. They are signals that provide information for human vital signs as well as images that depict physiological and functional characteristics of internal organs and tissues. From the perspective of the HPC domain, they could be divided in time-series signals as well as images. Typical examples of time-series signals are the ECG and EEG signals, that are records of the electrical activity of the heart and brain respectively. Xrays, CTs, MRIs, fMRIs, SPECT are only a few of different views of human organs and their functionalities. Both these type of data require advanced signal and image processing techniques to perform tasks that assist in disease diagnosis and treatment plans as well as research and clinical trials. Most common tasks include pattern recognition and classification of findings within the signals and images as well as prediction of outcomes. This thesis focuses on a single classification application that processes ECG signals and detects all instances of arrhythmia and therefore abnormal

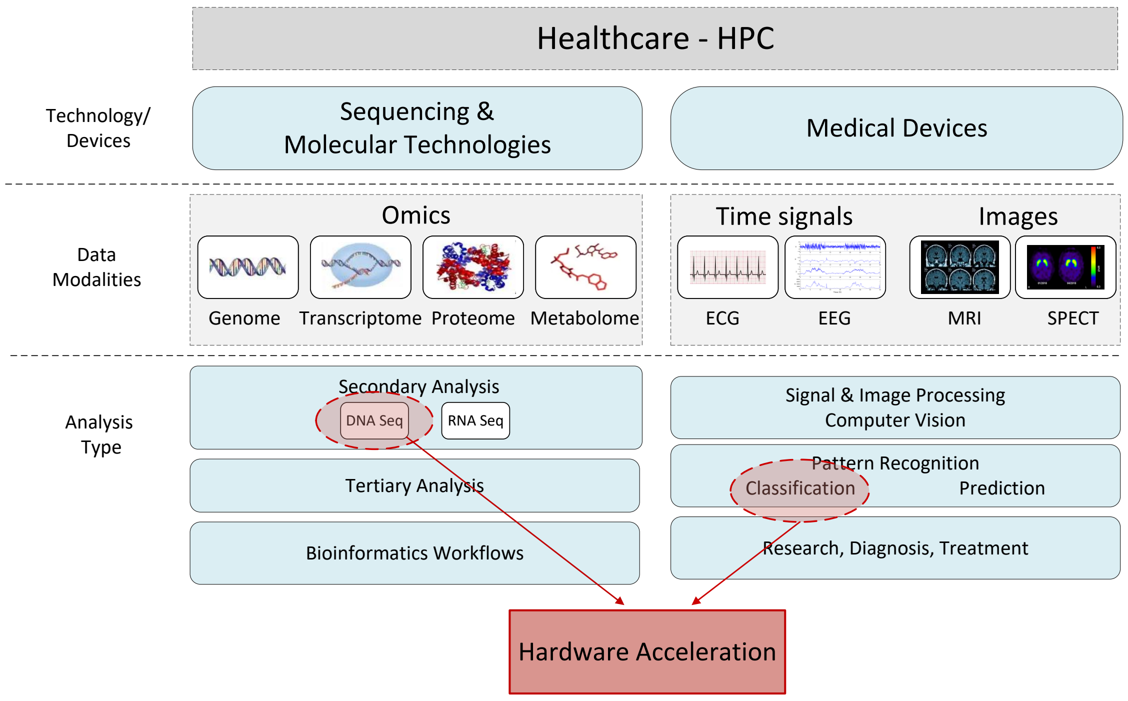


Figure 2.1.: Thesis Scope and Contribution within Healthcare HPC Domain.

heartbeats.

The scope of the thesis is to employ techniques from the High Performance Computing domain in order to optimize the target applications. The basic tool we utilize to deliver efficient solutions is *High Level Synthesis* programming targeting FPGA devices. In the next paragraphs, we present a high-level overview of the challenges encountered by related works and the solutions leveraged to address them. Based on this analysis, we propose our own implementations, that aim to build upon the existing solutions and suggest potential improvements. For each application, we present open problems of prior art and enumerate the contributions of each of the proposed frameworks.

2.1. Challenges in HLS optimization techniques and Thesis Contributions

The first part of the thesis examines hardware acceleration of an ECG arrhythmia detection application performed through a machine learning model. This section presents the most common challenges that designers encounter when leveraging High Level Synthesis tools for FPGA acceleration of compute-intensive applications such as machine learning models. Applying optimization techniques that target memory and efficient design space exploration have been broadly leveraged thanks to their high impact in performance.

Memory optimization techniques: An effective strategy for optimizing an application in HLS tools is leveraging the memory access patterns of the algorithms and tuning the memory layout and hierarchy to efficiently support them. Towards this goal, there are several works that either propose a generic framework or a custom solution for a specific algorithm.

In [97] a methodology for automated memory partitioning is presented enabling parallel computation units to efficiently access multiple independent memory banks. The Z-polyhedral model for program analysis is used to address bank mapping and minimization of the total amount of memory required for the partitioned banks. In [98], authors perform memory profiling of various applications and split a single data structure into different memory banks for data parallelism to address the memory bottleneck problem. In [99], authors develop a micro-architecture with multiple memory systems, each one customized to allow parallel access to elements of one array. These systems manipulate an incoming stream of array elements so that those corresponding to parallel array references are led to different memory banks of the computation kernel.

In [100], [101] an exploration algorithm pinpoints the partitioning of an array so that elements accessed in parallel are assigned to different array partitions. Memory partitioning is combined with memory access scheduling in a cycle-accurate way. Authors in [102] target computational kernels at which parallel access to coefficient matrix takes place. They propose that reusable data can be cached to on-chip registers which are organized as chains to enable data caching with no need of extra control logic. Memory partitioning is then performed on the non-reusable data employing a padding technique which minimizes storage overheads when zeros are present in the partitioned vectors.

Design space exploration on HLS tools built-in optimization techniques: Despite its potential and ease of use, utilizing HLS in a straightforward manner usually delivers highly sub-optimal design solutions mainly due to the extremely large parameter space that has to be explored. Several researchers have identified these HLS inefficiencies [103–106], proposing the usage of design space exploration techniques to better guide accelerator synthesis. In [103] the authors exploit response surface models (RSMs) and spectral analysis for predicting the quality of the design points without resorting to costly architectural synthesis procedures whereas in [106] they combine compiler- and architectural-level transformations to create a solution space and guide the search using a gradient-based heuristic pruning technique. Authors in [105] propose a method to accelerate the Design Space Exploration (DSE) of behavioral descriptions for high-level synthesis based on a divide and conquer method whereas authors in [104] leverage learning-based methods. Most of these works do not take into account the structure of the targeted algorithmic description as opposed to a recent shift of attention to more targeted HLS optimization techniques tailored to the specific structural characteristics of the algorithmic descriptions, e.g. stencil computations [99–101].

Loop transformation optimization techniques: In [107], authors extract parallelism from Iterative Stencil Loop algorithms by performing data analysis and then design space exploration regarding different architectures. The main block of these architectures is a computational unit that computes all values of a selected data region at an iteration, taking into account data dependencies from the analysis and utilizing data values from a variable depth of proceeding iterations. An architecture instance comprises of many such units, each one working on a different data segment of the initial domain at the same time. In [108], authors extend an existing partitioning and scheduling algorithm [100] to also allow concurrent access to memory references across different loop iterations. Nested loops are also targeted in [109], where the unroll factor of each loop and the presence or lack of dataflow directive is determined to optimize the throughput vs area tradeoff of the produced accelerator. When the dataflow directive is applied, it enables the concurrent execution of the loops and the authors focus on minimizing the initiation interval of the kernel by iteratively optimizing the longest loop through exploration of various unroll factors.

In [110] the authors propose a technique to optimize on-chip memory allocation using loop transformations. Data reuse buffers are used to save data accessed by consecutive memory references. Loop transformation is used to reduce the buffer size by improving data locality of array accesses. Authors in [111] examine the adverse effects of typical memory partitioning techniques and especially bank switching and propose an exploration of loop unrolling to identify an unroll factor that alleviates the problem.

Thesis contributions: In this work we leverage the aforementioned types of techniques and combine them into a novel approach to accelerate an SVM classifier. The proposed optimization strategy combines custom source code rewriting techniques and automatic knobs of the HLS tool, and applies them in a stratified manner to deliver efficient delay-area SVM design solutions. In addition to this, the framework also proposes a design space exploration and pruning methodology to quickly pinpoint the pareto front. This results to very efficient SVM accelerator implementations, reporting 98.78% latency gains in comparison to design solutions derived by only leveraging the tools' capabilities, and to a co-designed ECG analysis and arrhythmia detection system deployed on Zynq SoC with a speedup of up to 43× in respect to a pure software implementation on an ARM processor.

The major contributions of our work are:

- We demonstrate the inefficiency of HLS tools' to fully exploit the available parallelism on its own and present source code structure optimizations that guide the tool to discover both data- and instruction- level parallelism. The source code structure in the first case assists the compiler to leverage coarse-grained parallelism. The second source code structure addresses inefficiencies regarding fine-grained parallelism, i.e. exploiting pipeline and parallelism within a single block of code.

- We implement an HLS exploration framework that thoroughly investigates the automatically available tuning knobs and quantifies the impact of each one. Based on this exploration, we deduce a set of pruning criteria that match the accelerator's memory architecture to its computation and memory access patterns. The exploration generates a highly compact design space that lies on the same delay-area Pareto- front as the exhaustive design space.

- The proposed framework is developed in a modular manner, i.e. not integrated within the HLS engine, in order to enable compatibility and portability with other HLS tools in a smoother manner.

- We further enhance the framework, by incorporating a steepest decent meta- heuristic optimizer that delivers a single highly optimised SVM design solution in a time efficient manner. This is often preferable by the end-user in order to avoid the evaluation of an entire albeit small search space to select a single configuration.

- We evaluate the framework over approximate instances of the SVM classifier, that leverage loop perforation and precision scaling approximation techniques. Thus we combine both manual source code optimizations and built-in tuning knobs of the HLS tool with approximate techniques.

2.2. Challenges in Short read alignment and Thesis Contributions

High performance computing techniques have been greatly leveraged for optimizing and accelerating genomic analysis. Secondary analysis of sequencing data, i.e. read alignment, is a crucial step for subsequent workflows, as it reconstructs the sample genome and compares it to the reference genome for variant detection. Several software aligners have emerged for efficient execution of this stage, such as BWAMEM [112], Bowtie2 [113], Edlib [114], WFA [115] and KWS2 [116]. Most aligners [112, 113, 117] adopt a *seed-and-extend* strategy to find possible matches of the read on the reference genome. *Seeding* fragments each read into even shorter pieces called *seeds* that align exactly on the reference genome and creates a pool of candidates for valid alignments. In the *seed extension*, each seed is extended into a gapped alignment, i.e. allowing mismatches or *edits*. In modern aligners, the extension step is most frequently implemented based on Smith-Waterman [118] dynamic programming algorithm for string matching.

The ever increasing size of input datasets and the computational intensity of the alignment step however exceeds the power of modern systems. Research efforts are opting to find effective acceleration techniques to meet the time requirements and increase the throughput of the alignment. The optimization efforts could be broadly categorized into two distinct trends based on the target of the optimization. The first approach includes solutions that operate in fact on the pre-alignment stage whereas the second one target the alignment stage itself. A common technique leveraged in both cases is offloading computations to powerful devices and co-processors. A plethora of works exploit hardware acceleration targeting different architectures, ranging from heterogeneous devices, e.g. FPGAs, to many-core accelerators such as GPUs or the Intel Xeon Phi. In this work, the focus lies primarily on accelerated designs that target FPGA devices.

Pre-alignment Optimizations: The major pre-alignment step is seeding, which requires the use of an index of the reference genome. Taking into account the size of the reference genome as well as the number of seeds, searching the genome needs to be implemented in an efficient manner. Typical approaches are optimizing the data structure for storing the genome index as well as the search algorithm for locating seeds through this index, e.g. FM-index [88, 89]. This guarantees that seed exact matches are found efficiently in terms of time and memory requirements. However, the volume of the seed matches that need to be extended remains. This leads to the creation of pre-filtering algorithms that aim at decreasing the vast amount of seed extension tasks, by discarding candidate alignments based on a predefined threshold for the edit distance. These candidates would most likely result in a prohibitive number of edits and would be eliminated. Several pre-filtering algorithms [90, 95, 119] have now been developed that have noted significant improvement in accuracy and performance since early efforts. Apart from the algorithmic innovation of these filters, most of them have also been implemented in hardware to accelerate the filtering time.

Alignment Optimizations: Despite the efforts made in the pre-alignment stage, the alignment itself remains at the centre of research interest. Most works focus on accelerating the core computation of alignment, which is a string matching algorithm. SmithWaterman [120] is one of the most popular ones. It is a dynamic programming algorithm that consists of two phases: (i) filling a similarity matrix and (ii) tracing back the similarity matrix to find the optimal alignment between the two sequences.

Custom reconfigurable processors for Smith-Waterman acceleration is an active field of research with several implementation solutions proposed [121]. Most Smith-Waterman accelerators [122], [123], [124], [125], [126] compute the similarity matrix based on a wavefront approach through a pipeline of PEs that forms a systolic array and computes a matrix anti-diagonal per time step. The authors in [124] provide a very detailed architecture as such, that implements a multistage-PE design, and optimize each stage in terms of resources utilization and delay. Similarly in [127], the authors propose a reconfigurable accelerator that implements a modified equation to improve mapping efficiency of a single PE, and a special floor plan to cut down the interface components routing delay. The authors in [128] employ a pipeline of PEs to calculate the similarity matrix but recalculate the matrices for traceback in software for highest-score alignments to avoid memory contention.

Although each of these publications suggest optimization techniques on Matrix-Fill, they do not provide a Traceback implementation. There are only a few works of accelerated sequence alignment based on Smith-Waterman with Traceback. The authors in [129] present an alignment engine that performs the traceback in parallel with the matrix fill stage with restrictions on sequence length due to on-chip memory bottleneck. On the contrary, the authors in [130] propose an alignment architecture that accelerates both the forward scan and traceback priotitizing space efficiency for variable reference lengths. The traceback procedure is implemented in software and the host initiates a full traceback by request. The traceback procedure is guided by the host by scheduling multiple partial traceback executions on hardware. Although this accommodates aligning sequences of variable length, it hinders integration with third-party aligners. Both accelerators are tailored for long sequence alignment with sequence lengths ranging from 50 to 16000 bases. These sequences are more than an order of magnitude longer compared to those required in short read alignment tasks. Although these works provide the traceback functionality under certain circumstances, they are not designed to cope with a high throughput input rate of short reads generated by an NGS platform.

There are also works that target acceleration of widely-used aligners or develop end-to-end hardware implementations of *custom aligners from scratch*. Authors in [131] emulate the Smith Waterman implemented in BWA-MEM for short-read alignment. The achieved acceleration is extracted from task-level parallelism rather than inner-task parallelism and the focus lies on the scheduling of parallel alignment tasks rather than the specifics of the integration and the handling of traceback intricacies. The work in [132] designs a hardware aligner from scratch based on the algorithm used by BFAST [133]. The implementation requires storing at least 20GB of memory for the genome and a table containing the candidate locations. No information is provided for the implementation of traceback stage.

Recently, Darwin [36] has proposed an end-to-end hardware acceleration for 3rd generation sequencing, implementing accelerators for both seed extract and extend phases and highlighting the importance of including the traceback step on chip so as not to undermine the benefits of hardware acceleration. For the extend phase, the authors introduce GACT algorithm which implements a modified SmithWaterman for arbitrarily long senquencesand is implemented in both ASIC and FPGA. The authors in GenAx [134] propose a highly efficient ASIC accelerator for both seeding and extension step that supports traceback and is based on a finite state automata instead of SmW. The same authors later propose SeedEx [135], an FPGA accelerator for the seed-extension step that targets a cloud FPGA and is also integrated in BWA-MEM aligner. GenASM [136] accelerator is an ASIC accelerator that performs the seed extension step based on Bitap algorithm and accelerates both short and long reads. The authors in [137] also present the ASAP

FPGA accelerator for short read alignment. ASAP introduces several modifications to the alignment procedure, that need extra validation before its adoption on existing NGS frameworks, e.g. it utilizes the Levenshtein distance computation rather than SmithWaterman. The latency of ASAP also is dependent on the number of mismatches between the short read and the reference and supports a simpler constant gap penalty model for the scoring scheme.

Thesis contributions: This thesis focuses on delivering a highly-efficient SmithWaterman accelerator that is optimized to accommodate integration within a realistic system. For that purpose we leverage dataflow computing to develop the accelerated design taking into account aspects such as system wide communication and accelerator call overheads which are often neglected. The proposed design methodology allows for integration into a widely-used open-source aligner and delivers up to $\times 2$ end-to-end speedup. We then further improve upon the design and leverage characteristics of the input dataset to deliver a resource efficient Banded SmithWaterman accelerator and a multi-dataflow accelerated system. The Banded Smitherwaterman accelerator achieves a speedup of $\times 1.53$ over the initial accelerator and $\times 3$ over state-of-the-art aligners whereas the multi-dataflow system delivers average speedups of $\times 1.8$ over the multi-instances version of the initial design.

In more detail, the major novelties and contributions of our work can be summarized as follows:

- We propose GANDAFL, a novel genome alignment architecture for SmithWaterman that is based on the dataflow computing model.
- We extend prior designs by implementing the complete Smith-Waterman algorithm along with the Traceback procedure, enriching current literature which is currently lacking in variety of detailed traceback descriptions.
- Within this architecture we propose a innovative data interleaving technique in order to further increase task-level parallelism and maximize the throughput via high utilization of the underlying FPGA.
- We integrate our accelerator in a real aligner by taking into account integration implications from the start and relying on a synergy of software/hardware co-design:
  - We diminish the data transfer overhead of integration by making the decision to move more computation on chip, i.e. implementing Traceback on hardware.
  - We diminish the accelerator call overhead by applying a radical software restructuring on the aligner source code that implements an aggregation-batching strategy in order to reduce the number of accelerator calls.

- We adopt a heuristic of Smith-Waterman, i.e. Banded Smith-Waterman, that only examines alignments with an upper limit for edits and exploit the inherent attributes to design a highly optimized dataflow implementation targeting FPGA devices.

- We bridge hardware acceleration and pre-filtering for alignment optimization by introducing a profile-driven design methodology leveraging the Banded Smith-Waterman accelerator. The methodology highlights the advantages of leveraging input-specific constraints to design resource-efficient accelerators that make up a highly parallel accelerated system.

- We implement a dataset-specific multi-dataflow system that significantly accelerates pre-filtering seed-extension alignment with negligible accuracy loss. The pre-filtering step classifies the input reads based on a lower limit for the number of edits and the seed-extension tasks are assigned to the suitable Banded Smith-Waterman accelerator based on the input edit threshold distribution.

Chapter 3.

Theoritical Background

In this chapter we present a theoritical background to the domains and tools discussed throughout the thesis. We explain the working principles and capabilities of the utilized tools so as to justify their selection and facilitate the understanding of our implementations. We also describe the basic concepts of the Genomics domain and present the structure and goal of typical pipelines of the field. The focus lies in providing details for short read alignment.

3.1. High Level Synthesis on Reconfigurable platforms

3.1.1. Basic Principles of High Level Synthesis

FPGA-based accelerators for computationally instensive algorithms are a popular and effective approach for optimizing and accelerating applications. Typically, FPGAs are programmed using Hardware Description Languages (HDL), which model the behavior and structure of logic circuits, e.g. VHDL and Verilog. Despite the immense capabilities of these languages, their use also translates into a great learning curve, inefficient development time and tiresome validation and verification procedures.

An alternative to programming in HDL is using High Level Synthesis tools (HLS) [138, 139]. A High Level Synthesis (HLS) tool is a design tool for generating application-specific IP from algorithmic C specification and thus allowing the designer to work at a higher level of abstraction, while creating high-performance hardware. It provides software developers with an easy way to accelerate the computationally intensive parts of their algorithms on a new compilation target, a Field Programmable Gate Array (FPGA). The FPGA provides a massively paralleled architecture with benefits in performance, cost and power over traditional processors. The main part of the application thus, is executed on the system's processor while a part of it is transformed into a Register Transfer Level (RTL) implementation that synthesizes into a FPGA.

The first step is to develop an algorithm usign a sofware language such as C, C++, SystemC, OpenCL. HLS then provides the ability of verification, which allows designers to validate the functional correctness of the algorithm faster than doing so in traditional hardware description languages. The next step is the synthesis of the software implementation into an RTL design. HLS offers the ability to control the synthesis process through optimization directives allowing the creation of specific high-performance hardware implementations. It performs some optimizations by default and also allows the user to impose directives and constraints of his own choice. The primary output is generated at the synthesis step and it is the implementation in RTL format. The RTL can be synthesized into a gate-level implementation and an FPGA bitstream file by logic synthesis. The RTL output is usually provided in the industry standard Hardware Description Language (HDL) formats of Verilog and VHDL. The RTL implementation can be packaged into an IP, so that it can be integrated into the selected hardware system [140].

HLS always begins with the compilation of the functional specification. This step transforms the input description into a formal model that exhibits the data and control dependencies through a Control and Data Flow Graph (CDFG) [141–143]. Allocation, scheduling and binding are the steps that follow and are the processes at the heart of High-Level Synthesis. Allocation defines the type and the number of hardware resources needed. Components can be added during the scheduling or the binding phase. During the scheduling process it is determined which operations will occur in which operation cycles. These operations can take place within one or several clock cycles, they can be chained or they can execute in parallel. The scheduling phase takes into account design, timing and user defined constraints. Binding is the process used that determines which hardware resource implements each scheduled operation. The decisions taken in the binding and allocation process influence the scheduling of the operations, thus resulting in these steps to be intertwined rather than happening in a serial fashion.

There are several tools and vendors that are based on these principles to create an optimized RTL implementation starting from a high level software language. Intel SDK Pro for OpenCL [144] is an Intel tool that transforms OpenCL source code into RTL descriptions targeting Intel FPGAs. Xilinx has developed throughout the years several tools from Vivado HLS [145] to Vitis HLS [146]. Legup [147] is also a very popular open-source high-level synthesis tool for FPGA-based processor/accelerator systems. The following paragraphs briefly present the frameworks utilized in this thesis.

3.1.2. Programming Models for Reconfigurable devices

Xilinx Vivado HLS

Vivado HLS allows the programmer to implement the algorithm in C, C++, SystemC, OpenCL. Once the code is written and the functional correctness is certified, the synthesis process takes place. The synthesis output is the RTL implementation of the algorithm. The RTL output is provided in the industry standard Hardware Description Language (HDL) formats of Verilog and VHDL. After synthesis, verification of the RTL implementation is possible, to ensure that the same results with the software implementation are being generated.

Vivado HLS can compile the C code into an implementation of high performance while maintaining an efficient resource usage through the HLS-defined pragma (directives), that are taken into account during the scheduling and binding process and result in an optimized IP block. High-Level Synthesis creates the most optimum implementation based on its own default behavior, the constraints, and the directives that the users specify. These optimization directives are selected so that the architecture created satisfies the desired performance and area goals. When synthesis completes, a synthesis report is generated. This report contains details on the performance metrics. After analyzing the report, optimization directives can be used to refine the implementation towards the desired outcome. In order to do that effectively, it is important to understand the metrics used to measure performance in a design created by HLS. The main ones are area, latency and initiation interval. The directives available from HLS aim at optimizing performance and area utilization. They can be applied on functions, loops, arrays and regions containing one or more of the above [2]:

- Functions: Directives applied to functions mainly aim at enabling two or more functions to execute concurrently and at removing function hierarchy in order to reduce function call overhead and examine logic optimization.

- Loops: Directives applied to loops can reduce the cost of transition cycles between different loops and between iterations of the same loop, improve latency, reduce resources and allow the parallel execution of multiple loops within a function.

- Arrays: Arrays are the basic construct to express memory in HLS and are implemented using block-RAMS. A block-RAM can be at most dual-port, which means that maximum 2 elements of the same array can be accessed at the same time. This introduces a bottleneck and prevents effective parallelization. The directives that are applied to arrays mainly address this issue. They change array layout by reshaping or partitioning to remove bottlenecks without requiring changes to the original code. There are also directives that map arrays together in order to reduce

Table 3.1.: Basic HLS Optimization Directives

Directive	Description
PIPELINE	Reduces the initiation interval by allowing the concurrent execution of operations within a loop or function.
DIRECTIVE	Enables task level pipelining, allowing functions and loops to execute concurrently. Used to minimize interval.
INLINE	Inlines a function, removing all function hierarchy. Used to enable logic optimization across function boundaries and improve latency/interval by reducing function call overhead.
ARRAY PARTITION	Partitions large arrays into multiple smaller arrays or into individual registers, to improve access to data and remove block-RAM bottlenecks.
ARRAY MAP	Combines multiple smaller arrays into a single large array to help reduce block-RAM resources.
ARRAY RESHAPE	Reshape an array from one with many elements to one with greater word-width. Useful for improving block-RAM accesses without using more block-RAM.

area.

In Table 3.1 some of the basic HLS directives are briefly descripted. These directives are the base for the design space exploration performed in Chapter 4.

Maxeler Dataflow Computing Model

A recent approach to programming FPGAs through High Level Synthesis tools has been introduced through the Maxeler Dataflow Computing Model. Maxeler has developed a programming language, called MaxJ, which is based on the dataflow computing model [148]. This language can describe the hardware that implements an initial software application and transform it to RTL through MaxCompiler.

An elaborate presentation of the maxeler dataflow model and the ideas introduced by dataflow computing can be found in [149]. The fundamental principles of the dataflow computing paradigm are different to the ones of the typical processors. In an instruction processor, instructions and data are read from memory into the processor core, which performs operations before returning the results to memory. If the result is required for the subsequent operation, it must be reloaded from memory, computed, and then written back to memory. The operating model is inherently sequential, and its performance is limited by the speed with which data can move round this path from memory, through an operation, and back to memory.

On the other hand in dataflow computing, the programmer constructs a circuit that comprises of functional units and represents the initial software application. Data is streamed from memory, onto the chip, and propagated from one unit to another. The data are not written on off-chip memory until the computation is finished. They are always on chip, available in FIFOs if needed for longer times. Unlike the CPU where operations are executed in subsequent points in time on the same hardware part, in dataflow the computation is laid out spatially on the chip. There are no instructions, instruction decode logic or branches. As data flows through the units, some tight dependencies are lifted, allowing for deep pipelines and high throughput.

The programmer writes code in the MaxJ language, which is a Java meta-program that describes the structure of the dataflow engine that should be created. The dataflow engine is the Maxeler naming adopted for the accelerator/kernel. The computationally intensive components of the application can be offloaded to one or more FPGA dataflow engines and the remaining application still runs on the conventional CPU unchanged. To create the dataflow engine for a particular application the programmer creates one or more Kernels and a Manager. The kernels contain the computation portion of the application. The Manager describes how the Kernel inputs and outputs are connected to the outside world and to other kernels.

Once the development is complete, the bitstream creation process follows the typical procedure of synthesis, placement and routing. MaxCompiler translates the Java into HDL language and then builds the bitstream by invoking typical vendor software depending on the target platform. For example, if the target is Intel FPGAs, MaxCompiler invokes the Intel/Altera flow whereas if the target is Xilinx FPGAs the Xilinx flow. The output of the building process is a chip configuration file (the .max file) that contains the bitstream to configure the FPGA device and also meta-information enabling it to be easily incorporated into a full application. The programmer then exchanges data between the software program and the dataflow engine using the MaxCompilerRT run-time library API.

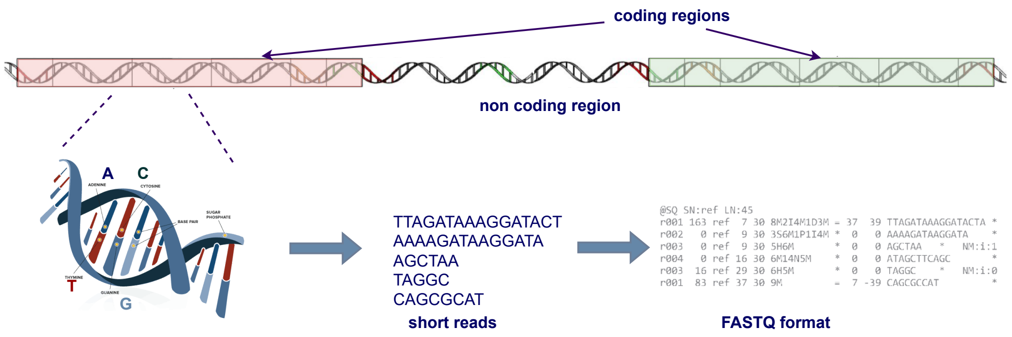
3.2. Bioinformatic Applications

3.2.1. DNA sequencing and Genomic Analysis

The Human Genome project [150] was an international joined research effort to determine the order of bases in the human genome DNA and create maps of the location of genes. Upon its successful completion, a reference genome for the human was available for the first time. This fact combined with advances in sequencing technologies [151] that produced millions of nucleotide sequences of an individual's DNA have paved the way for the high-throughput sequencing era and the establishment of the field of bioinformatics. Bioinformatics [152] is an interdisciplinary field as it requires the expertise of multiple domains such as biology, physics, computer science and mathematics. The essential incentive and goal of the field is to provide a management scheme and storage for reference genome data as well as new entries. Efficient data curation accommodates achieving the second goal, which is to develop tools and frameworks that analyse data and annotate the results of the analysis with biological meaning.

The principal type of data handled by bioinformatics application is raw genome/DNA data. A genome is essentially a long string of nucleotide bases: adenine, cytosine, guanine and thymine (A, C, G and T respectively). An organism's complete set of genetic instructions and information is enclosed in regions of the genome, called genes. Fig.3.1 illustrates the DNA structure and how this comprises of coding regions (i.e. regions that are translated in protein sequences and determine cellular functions) and non-coding regions (i.e. non translatable repetitive sequences). Advances in sequencing technologies [153] have achieved a significant decrease in the cost and time required to sequence the DNA sample of an individual, which has led in an explosion in available sequenced data. The output of sequencing consists of sets of relatively short genomic sequences, usually referred to as reads. Examples of short reads that have originated from a DNA sample can be seen in Fig.3.1. As the technologies for sequencing evolve, so does important parameters such as read throughput, length and error rates. Sequencing technologies evolved from First to Second and lastly Next Generation [154]. First generation platforms produced reads with average read length 400-900 bases and a very low throughput per run [155]. Second generation sequencing platforms [156] (eg Illumina/Solexa, SOLiD [157], [158],) generate many millions of nucleotide short reads, with lengths that range primarily between 50 to 300 bp (base pairs). The error rate ranges from 0.1 to 1% thanks to short length and high coverage. The latest technologies (eg Pac Bio, Oxford Nanopore [159], [160]) generate long reads (thousand of bases long), as it has been discovered that they assist to discover previously undetectable structural variants. This however comes at the cost of high error rates, up to 20%. Depending on the end goal of each study, the most suitable platform and type of reads are selected.

The reads generated by sequencing platforms can then be used as input to several clinical



```
TTAGATAAAGGATACT
AAAAGATAAGGATA
AGCTAA
TAGGC
CAGCGCAT
```

```
@SQ SN:ref LN:45
r001 163 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTA *
r002   0 ref  9 30 3S6M1P1I4M * 0    0 AAAAGATAAGGATA    *
r003   0 ref  9 30 5H6M       * 0    0 AGCTAA   *   NM:i:1
r004   0 ref 16 30 6M14N5M    * 0    0 ATAGCTTCAGC       *
r003  16 ref 29 30 6H5M       * 0    0 TAGGC    *   NM:i:0
r001  83 ref 37 30 9M         = 7  -39 CAGCGCCAT         *
```

Figure 3.1.: Example of an Alignment between a reference and read sequence. It includes all possible types of mismatches among DNA bases.

applications that leverage bioinformatics frameworks. Genotyping [161] is one of the most popular genomics application, that allows scientists to explore genetic variants such as single nucleotide variants, copy number variants, and large structural changes in DNA. Different expression of genes among individuals create the genetic diversity and lead to a variety of phenotypes. Genome study is essential to discover the mechanisms that lead to this diversity and is of great value to emerging fields like personalized medicine and research for various often incurable genetic diseases. Genome applications that fulfil this purpose are variant calling [162], differential gene expression [163], phylogeny creation [164] etc. Gene Expression and Transcriptome Analysis [165] are also critical applications that allow scientists to study the mechanisms of translating genes and gain a deeper understanding of biology. Apart from genetic studies, genomic tools also accommodate the epigenetics study [166], which examines how environmental factors can alter how genes work.

3.2.2. Short read alignment

This work focuses on applications that stem primarily from the genotyping subdomain using Second generation sequencing technology. Fig.3.2 illustrates a typical variant-discovery workflow and the most important steps of the workflow. The first step focuses on the generation of the data required for these workflows. A selected sequencing technology reads the sequence of nucleotide bases in a DNA molecule of an individual and converts these raw signals into short fragments of bases, called short reads. During the generation of short reads, unique sequencing errors and biases are introduced and therefore quality checks are required to identify and correct them. The resulted short reads data are saved in FASTQ format [167] (also illustrated in Fig.3.1).

The next steps. i.e. short read alignment and variant calling, facilitate the reconstruction of the genome of the sample and comparing it to the reference genome of the organism. Short read alignment performs the mapping of the short reads generated in the first step

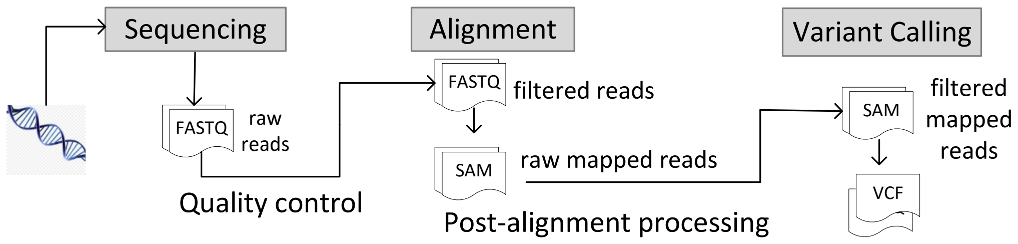


Figure 3.2.: Typical Genomic Pipeline stages. Sequencing generates short reads from a DNA sample and is followed by aligning them to a reference and calling variants.

to a location in the reference genome that is most likely its origin. Both reference and read sequences are encoded using numerical values, as illustrated in Fig.3.3. The short read alignment process results in alignments that can be either perfect matches to the corresponding reference location or include mismatches. Fig.3.3 depicts an example of an alignment between a reference and read DNA sequence and demonstrates some of the most frequent variants in the sequence of bases that can potentially lead to genetic mutations. Gaps are used to account for insertions or deletions in the sequence, i.e. reference gap and read gap respectively. After aligning, a Sequence Alignment Map (SAM) file is produced, which includes information on the alignments of reads [168]. A post-alignment processing step is also invoked to correct technical biases, and the corrected alignments are ready for variant calling analysis. Finally the variant calling step identifies differences between the sequencing reads and the reference genome. It is worth mentioning that, localized realignment is also performed during the variant calling stage to correct artifacts introduced during the alignment phase. The pinpointed variations are reported in an output file in the Variant Call Format (VCF) [169].

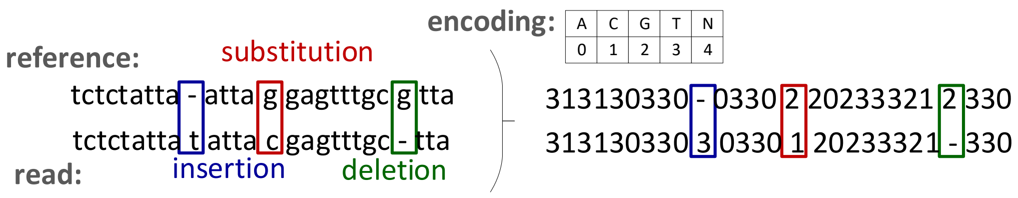


Figure 3.3.: Example of an Alignment between a reference and read sequence. It includes all possible types of mismatches among DNA bases.

As shown, short read alignment is at the heart of the genomic pipelines. Several software tools have been developed therefore to perform alignment of shorts reads to the reference genome. Bowtie [170, 171], Bowtie2 [113], Soap2 [172], BWA-MEM [112] are some of the widely used ones. An exhaustive review and categorization of available aligners has been reported here [173]. A first categorization takes place based on the type of index used to perform quick and efficient queries on the genome reference. The efficiency is based on the principle of producing a minimal memory footprint by storing the redundant subsequences of the reference genome only once. Each aligner creates a pre-built index from the reference genome that allows for quick search of substrings across its chromosomes.

The most popular indexing technique is hashing, thanks to easy implementation, small indexing time and fast seed query speed [173]. The second most popular index is the suffix-tree-based index. Due to difficulties in implemented the suffix tree, most aligner tools rely on the BWT transform [174] and the FM-index [88], to emulate the suffix-tree traversal.

The aligners operate on a set of sequencing read files in FASTQ format and output a set of alignments in SAM format [168]. For each read, they apply a specific alignment strategy to determine the minimum number of differences between the sequences, the type of differences and their exact location on the DNA strands. There are two major types of algorithms that perform this task, Dynamic Programming (DP) based ones and non-DP based ones. The latter ones are represented mainly by the Hamming distance [175] and the Rabin-Karp algorithm [176]. However the DP-based ones are the most popular and mainly include the Smith-Waterman [120] and the Needleman-Wunsch [177] string-matching algorithms.

Each resulting alignment has a score that represents the probability for the read to originate from the corresponding reference's location in the genome. The alignment result reported follows a specific format when reported, e.g. CIGAR [168], that contains information on the read sequence, which strand of reference genome aligns against, alignment score, number of differences, type of differences, location of differences on the reference sequence etc. Each aligner usually provides a plethora of reporting protocols for the found alignment (e.g. report highest score, report k first alignments, report all alignments etc) and allows the user to specifically select the suitable one for the current needs.

Chapter 4.

An Exploration Framework for Efficient High-Level Synthesis of Support Vector Machines: Case Study on ECG Arrhythmia Detection for Xilinx Zynq SoC

In recent years, Support Vector Machine (SVM) classifiers have played a crucial role in providing data fusion and high accuracy classification solutions for various, complex, non-linear problems. Its popularity accompanied by the ever-increasing need of implementing it on computationally weak, portable or even wearable systems has refuelled the effort to accelerate its execution. In this chapter, we explore FPGA-based acceleration to produce efficient SVM hardware co-processors. We propose a systematic two-level approach for SVM acceleration, which first optimizes the global structure of the original SVM's behavioral description to exploit the data- and instruction-level parallelism and then further refines it through a targeted design exploration that matches the accelerator's memory architecture to its computation and memory access patterns. The proposed methodology has been implemented as a framework on top of the state-of-art Vivado High-Level Synthesis (HLS) tool. We evaluate the effectiveness of the methodology through a rich set of analysis and validation results which show that its adoption delivers SVM accelerator designs achieving latency gains of up to 98.78% in respect to Vivado-HLS default optimized solution. Finally, using as a case study an ECG analysis and Arrhythmia detection system we show that a target Zynq programmable SoC utilizing the optimized SVM accelerator design outperforms pure software implementations in numerous single or dual core target platforms, achieving speedups which range from ×10 up to ×78. This chapter is based on our publications in [178, 179]. The work is the result of collaboration and is also included in Ph.D. Thesis "Design Methodologies for Resource Management of Many-core Embedded Systems" [180]. The author in [180] contributed with the software part of the application, through the development of the ECG Analysis Flow and the software evaluation on several embedded devices. The author of the current thesis developed the SVM hardware accelerators through Vivado HLS as well as the optimization methodology and guidelines for efficient design and synthesis.

4.1. Introduction

Computer systems have nowadays dominated numerous aspects of human life and in the advent of Internet of Things era, the vision is that a wave of portable sensor based devices will constantly sense and monitor user and environmental activities. This new architectural paradigm is inherently in need of a fast and efficient way to process the acquired data both on site and remotely in order to produce knowledge that will aid the actions of the user. Towards this direction, machine learning tools are in the spotlight for providing the necessary algorithmic infrastructure that will meet the data fusion requirements of this new architecture. Nevertheless, the envisaged applications are also bound by execution latency requirements since they operate under real-time constraints. Consequently, there is an emerging need for accelerated execution of various machine learning based computation kernels. Taking all these into account, this work aims at providing a design methodology for the acceleration of Support Vector Machine classifiers using High Level Synthesis and targeting FPGA based systems.

SVM-based classifiers [181] have grown very popular as the key element of machine learning based applications due to their capability of accurate predictions. They have been utilized in fields such as text recognition [182, 183], bio-medical applications [184, 185], image processing [186–188] and more recently activity recognition in mobile devices [189] and deep learning [190]. The popularity of these classifiers is twofold. On the one hand they provide very high classification accuracy even in problems which exhibit complex non-linear distribution in their extracted features space. On the other hand, their structure, based on stencil computation operations forms a promising candidate for applying acceleration techniques [191, 192].

In this paper, we focus on FPGA-based SVM acceleration and to showcase the effectiveness of our derived accelerators we utilize an ECG-based arrhythmia detection flow as a case study. In [193], where Electrocardiogram analysis is performed, it has been shown that when the classification problem becomes very complex, i.e. the computational requirements of the SVM classifier are increased, hardware acceleration can be the key for meeting the time constraints of the application and result to power gains.is of paramount importance Approximate computing can also be leveraged to accomplish this goal. Approximate computing, exploits the inherent error resilience of many application domains to trade accuracy for gains in other metrics (e.g., performance, energy) and manages to be established as an alternative for efficient systems design [194]. Driven by this high potential for performance and utilization efficiency, designing approximate circuits has attracted significant research interest. In hardware design, algorithmic and logic approximations are applied [194–200]. Considering that signal/image processing applications and neural networks (such as classifiers) are perfect candidates for both FPGA design and approximation, we examine the individual as well as combined impact of hardware and approximation techniques to accelerate SVM classifiers.

SVM accelerators are implemented utilizing Vivado-HLS, a state-of-art HLS solution of industrial strength, that enables fast exploration and prototyping of different architectural design decisions. Despite its high productivity, utilizing HLS in a straightforward manner usually delivers highly sub-optimal design solutions mainly due to the extremely large parameter space that has to be explored. Several researchers have identified these HLS inefficiencies [103–106], proposing the usage of design space exploration techniques to better guide accelerator synthesis. Similarly, there is research potential in the approximate computing field. Approximate design mainly targets arithmetic units (e.g., adders [194, 195] and multipliers [197, 198]), but their efficient application in complex accelerator circuits is not comprehensively analyzed and remains arguable. Despite significant results for approximate accelerators, e.g., [199–201], research activities on approximate FPGA accelerators are still limited.

Within this context, we propose a two-level systematic design framework for HLS which aims at providing optimized SVM HW accelerator design implemented on FPGA based systems. Followingly, we further apply this strategy on approximate instances of the utilized SVM. In brief, the main contributions of this work are summarised as follows:

- We present HLS source code structure optimization techniques to fully exploit the data- and instruction-level parallelism of the SVM classifier (1st Level of Framework Optimization). We show even mature HLS tools fail at fully exploiting the inherent parallelization features of the algorithm and provide comparative results against the proposed technique.

- We extensively investigate and quantify the impact of HLS directives on design objectives/metrics of the resulted SVM HW accelerator. This analysis is translated to a set of design space pruning guidelines which are experimentally validated to provide a highly compact design space which lies on the same delay-area Pareto-front as the full one.

- We incorporate the pruning guidelines as the key element of the automated HLS directives exploration tool that matches the accelerator's memory architecture to its computation and memory access patterns (2nd Level of Framework optimization). We further enhance the framework, by incorporating a steepest decent meta-heuristic optimizer that delivers a highly optimised SVM design solution in a time efficient manner.

- We show that the HW accelerator derived by our two-level optimization strategy manages to outperform a large number of other SVM implementations including high-end dual-core embedded systems.

- Lastly, we present an approximate FPGA-based SVM accelerator developed in Vivado HLS that is further enhanced by applying the aforementioned framework.

In more detail, extensive experimentation shows that the adoption of the proposed HLS design methodology leads to very efficient SVM accelerator implementations, reporting 98.78% and 89.26% latency gains in comparison to the design solutions derived by Vivado-HLS and advanced optimization meta-heuristics, respectively. The derived design configurations have been further evaluated on SVM with very large support vector sizes, showing the sustainability of the resulted design configurations for SVM accelerators at scale. Finally, the derived SVM accelerator has been successfully incorporated in a co-designed ECG analysis and arrhythmia detection system deployed on Zynq SoC, delivering speedups of up to 43× and 78× in respect to a pure software implementation on ARM processor and a co-designed solution utilizing SVM accelerator derived directly by Vivado-HLS. The approximate computing approach also shows great promise with the classification accuracy ranging from 90% to 99% and speedup from 1× to 18×.

The rest of the paper is organised as follows. Section 4.2 provides a literature review on FPGA-based ECG analysis acceleration as well as a short overview of HLS design exploration and optimization techniques. Section 4.3 introduces the structure of the SVM classifier and describes the characteristics of the ECG based case study application. Section 4.4 summarizes the key points of the proposed design exploration and optimization methodology for efficient SVM accelerator synthesis as well as presents the synergy of this methodology with approximation techniques. Section 4.6 describes the experimental setup and provides the experimental results regarding the efficiency of the exploration strategy and the derived HW designs while Section **??** concludes the paper.

4.2. Related Work

High Level Synthesis provides a transparent synthesis flow for designing hardware from abstract algorithmic descriptions. It enables designers to perform fast RTL prototyping and architectural exploration of their design solutions and it is extensively utilized for providing acceleration of computationally intensive kernels such as stencil computations. Stencil computations are found in several applications' algorithmic descriptions, e.g. SVM classifiers, exhibiting memory bottlenecks since they require the simultaneous manipulation of multiple elements of the same array thus failing to meet the throughput demands.

Several research works have proposed general design exploration strategies for HLS-based architectural optimization, to alleviate such problems. The proposed exploration framework forms an automated tool-flow solution already fully integrated with Vivado-HLS. Implementing the exploration framework in modular manner, i.e. not integrating the exploration engine inside the HLS engine (in any case not possible with Vivado-HLS which is not an open source tool) is strategic design decision that enables the proposed framework to be retargeted and integrated with other HLS tools in a smoother manner. This type of modularity, i.e. treating the HLS engine as an external tool, has also been adopted by other state-of-art design space exploration frameworks [103–106, 193] targeting the HLS domain. Most of these works have proposed exploration strategies without however taking into account the structure of the targeted algorithmic description [103–106]. Recently, a lot of attention has been shifted towards more targeted HLS optimization techniques tailored to the specific structural characteristics of the algorithmic descriptions, e.g. stencil computations [99–101].

In [97] a methodology for automated memory partitioning is presented enabling parallel computation units to efficiently access multiple independent memory banks. The Z-polyhedral model for program analysis is used to address bank mapping and minimization of the total amount of memory required for the partitioned banks. In [98], authors perform memory profiling of various applications and split a single data structure into different memory banks for data parallelism to address the memory bottleneck problem. In [99], authors develop a micro-architecture with multiple memory systems, each one customized to allow parallel access to elements of one array. These systems manipulate an incoming stream of array elements so that those corresponding to parallel array references are led to different memory banks of the computation kernel. Similar to this principle, we focus on one computation kernel and partition or reshape arrays according to the access pattern of the algorithm so that the number of memory banks or their width is sufficient for the required parallelism.

In [100], [101] an exploration algorithm pinpoints the partitioning of an array so that elements accessed in parallel are assigned to different array partitions. Memory partitioning is combined with memory access scheduling in a cycle-accurate way. Authors in [102] target computational kernels at which parallel access to coefficient matrix takes place. They propose that reusable data can be cached to on-chip registers which are organized as chains to enable data caching with no need of extra control logic. Memory partitioning is then performed on the non-reusable data employing a padding technique which minimizes storage overheads when zeros are present in the partitioned vectors. Our proposed techniques are closer to these approaches, since we also partition each array according to the access pattern in each loop iteration to allow concurrent access. We further examine grouping these elements in elements of greater word-width and also combine the two techniques. As opposed to utilizing one-dimensional arrays and perfect loop nests, we apply our methodology to both one and two-dimensional arrays and on an imperfect loop nest.

In [107], authors extract parallelism from Iterative Stencil Loop algorithms by performing data analysis and then design space exploration regarding different architectures. The main block of these architectures is basically a computational unit that computes all values of a selected data region at an iteration, taking into account data dependencies from the analysis and utilizing data values from a variable depth of proceeding iterations. An architecture instance comprises of many such units, each one working on a different data segment of the initial domain at the same time. In [108], authors extend an existing partitioning and scheduling algorithm [100] to also allow concurrent access to memory references across different loop iterations. Similar to this idea, we utilize partitioning of the initial data set to enable parallel execution of our computational kernel on the segments of the initial set and thus allow concurrent execution of loop iterations. Nested loops are also targeted in [109], where the unroll factor of each loop and the presence or lack of dataflow directive is determined to optimize the throughput vs area tradeoff of the produced accelerator. When the dataflow directive is applied, it enables the concurrent execution of the loops and the authors focus on minimizing the initiation interval of the kernel by iteratively optimizing the longest loop through exploration of various unroll factors. We also apply the dataflow directive to allow instances of the SVM kernel, which are basically nested loops, to execute in parallel. The instances are optimized using our proposed design space exploration, that includes extensive research of loop unrolling.

In [110] the authors propose a technique to optimize on-chip memory allocation using loop transformations. Data reuse buffers are used to save data accessed by consecutive memory references. Loop transformation is used to reduce the buffer size by improving data locality of array accesses. Authors in [111] examine the adverse effects of typical memory partitioning techniques and especially bank switching and propose an exploration of loop unrolling to identify an unroll factor that alleviates the problem. Loop unrolling technique is extensively utilized in our work and is combined with partitioning techniques in order to match the array layout to the algorithm access pattern and address memory burst issues and bottlenecks.

Therefore, in this work we propose a novel optimization strategy, implemented on top of the state-of-art Vivado-HLS tool, that combines data-, instruction-level parallelism and memory architecture customization to deliver efficient delay-area SVM design solutions. An additional value to our work is attributed to the fact that we do not only apply manual restructuring techniques and partitioning but further enhance these results by exploring the tuning knobs of the employed tools.

4.3. Support Vectors Machines based Classifier

4.3.1. Analysis of SVM classifier

Support Vector Machines (SVMs) are supervised machine learning models used for data-driven modelling and classification. The classifier is trained against a set of feature vectors composed of attributes of the points under classification accompanied by their respective class labels. Without loss of generality, for the context of this work we will consider the possible classes to be identified by 1,-1. The training phase of the algorithm is performed offline and is usually a computationally intensive task.

The outcome of the training phase is a set of support vectors, i.e. critical points of the different classes that define the classification hyper-planes. A new input vector is classified according to its distance from the support vectors. A kernel function K is used to map input vectors to a space where different classes are linearly separable. In total, having trained an SVM model of N_sv support vectors, the function for classifying an input feature vector $\mathbf{x}$ is of the following form:

$$Class = sgn(\sum_{i=1}^{N\_sv} (y_i * a_i * K(\mathbf{x}, \mathbf{sup\_vector}_i)) - b) \tag{4.1}$$

where K is the kernel function, $\mathbf{sup\_vector}_i$ is the i-th support vector and y_i, a_i are values derived from the training process. Coefficient b is a bias value, also a result of the training process and is constant over all support vectors.

The kernel function is very important for the accurate prediction of input data. In this work, Radial Basis Function (RBF) is the chosen kernel since biomedical applications have proved to perform poorly when linear kernels are used, since medical data are characterized by non linear relations of their attributes. The advantage of the RBF kernel over other non linear kernels is that RBF has fewer parameters and fewer numerical difficulties. The RBF kernel for two vectors a and b is defined as:

$$K(\mathbf{a}, \mathbf{b}) = exp(-\gamma||\mathbf{a} - \mathbf{b}||^2) \tag{4.2}$$

Listing 4.1: SVM original prediction code

```
const float sv_coef[N_sv];
const float sup_vectors[D_sv][N_sv];

void SVM_predict (float test_vector[D_sv],
        int * y) {

        loop_i:for (i=0; i<N_sv; i++){
                loop_j:for (j=0; j<D_sv; j++){
                        diff=test_vector[j]-sup_vectors[j][i];
                        norma = norma + diff*diff;
                }
                sum = sum + exp(-gamma*norma)*sv_coef[i];
                norma=0;
        }

        sum = sum - b;

        if (sum<0)
                *y = -1;
        else
                *y = 1;
}
```

Listing 4.1, introduces the C-language based implementation of equation 4.1 and this code will be used as the basis for the implementation of our HLS based HW accelerator. The input of the code is a new feature vector to be classified and its outcome is a label of -1,1 which classifies the input vector in one of the two available classes. Table 4.7 describes all variables declared in Listing 4.1 and what they stand for. The number of the support vectors *N_sv* and the length of the feature vector *D_sv* along with the kernel selected have great impact on the complexity of the classifier.

4.3.2. Use Case: ECG-based Arrhythmia Detection

Electrocardiogram (ECG) is a fundamental biological signal for health status monitoring and assessment due to its inherent relation to heart physiology [202]. Consequently, its analysis and interpretation has been established as an important field in modern medicine and this has spawned various inter-disciplinary studies towards its digital processing. Recently, machine learning techniques have dominated ECG analysis given the complexity to derive accurate models in the effort of assessing and predicting the state of the heart [203, 204].

Table 4.1.: Declaration of variables in Listing 1

variable	description
N_sv	number of support vectors
D_sv	dimension/features of support vector
sv_coef	array of coefficients for each support vector
$sup\_vectors$	array of support vectors
$test\_vector$	feature vector of heartbeat
$diff$	difference between elements of vectors
$norma$	squared euclidean distance of vectors
$gamma$, b	parameters γ and b of RBF kernel product of y_i, α_i of equation 4.1
sum	accumulator for contribution of each support vector
y	pointer to classification result

Arrhythmia, i.e irregular heartbeat, is considered as one of the most commonly encountered heart malfunctions. Due to this critical condition the field of detecting signs of arrhythmia is very important and automated diagnosis flows are a valuable tool in the hands of medical experts. In this work, we use as a case study arrhythmia detection based on ECG signal analysis. Detection is performed through SVM-based classification, which has been shown to exhibit high accuracy in detecting problematic beat patterns [193, 203].

Feature vectors are extracted from a heart beat and its produced classification label indicates whether the beat is normal or abnormal in terms of arrhythmia signs. To train and test the SVM prediction model, we utilized MIT-BIH Arrhythmia Database [205], a combined effort of MIT and Beth Israel Deaconess Medical Center - which is composed of 48 fully annotated half-hour two-lead ECG signals with the collaboration of patients with different medical files and physiology characteristics.

Fig. 4.1 depicts an overview of the process of analysing an ECG signal. The main processing stages are:

- **Noise removal** that filters the signal using a band-pass filter to remove artifacts resulting from patient breathing and movement or noise imported by the power line.

- **R peak detection and heart beat segmentation** that detects a heart beat inside the acquired ECG signal data. If an R peak is detected then a new heart beat has been located and its data are segmented for further processing.

- **Feature extraction process** is imposed on the heart beat in order to extract its

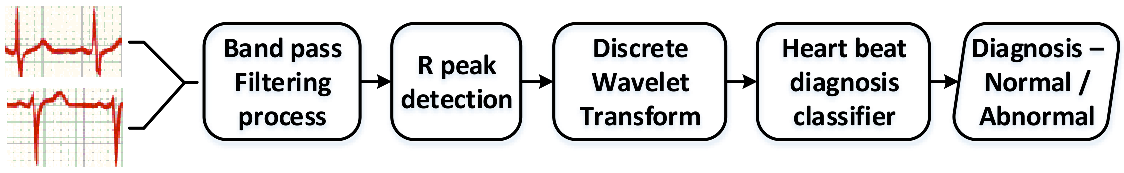

Band pass Filtering process
R peak detection
Discrete Wavelet Transform
Heart beat diagnosis classifier
Diagnosis – Normal / Abnormal

Figure 4.1.: Utilized ECG analysis flow

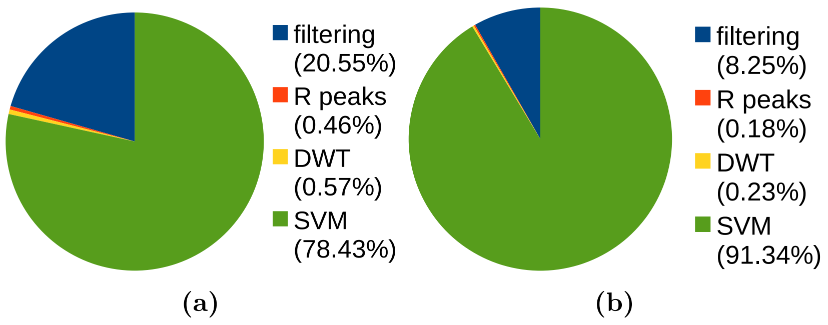


Figure 4.2.: Average CPU utilization per heart beat processing (a) SVM models of moderate computational requirements, (b) SVM models of high computational requirements

characteristics. In this work, we utilize Discrete Wavelet Transformation [203] as the core of the feature extraction process.

- **Diagnosis classification**: that performs the actual detection using SVM classifier and concludes whether the heart beat exhibits arrhythmia signs or not.

Fig. 4.2 illustrates the average CPU utilization per heart beat processing for the different processing stages of the ECG analysis flow executed on an Intel Quark SoC [206]. Different models of SVM classifiers, with increasing computational requirements (in terms of number of support vectors and input vector size) were used during the profiling phase. In all cases the inputs of the device are signals derived from MIT-BIH arrhythmia database. Fig. 4.2a shows that even for moderate complexity SVM models, execution of the classifier dominates the required CPU time. This is emphasized in the case of high complexity SVM models (Fig. 4.2b) where SVM execution takes up in average more than 90% of the CPU time needed for detecting arrhythmia in a single beat.

In this paper, a HW/SW co-design paradigm is used, to instantiate the ECG analysis flow in a combined CPU-FPGA system. The CPU is occupied with system management and pre-diagnosis ECG processing while the actual diagnosis is executed on a HW accelerator instantiated on the FPGA fabric. The SVM model used in the exploration process was selected according to the methodology presented in [207] so that it provides the highest classification accuracy. Additionally, its size is sufficient enough to allow all HLS optimizations to be applied on it and still have enough FPGA resources to instantiate the accelerator on a Zedboard [208].

Table 4.2.: Utilized SVM model parameters

parameter	meaning	value
N_sv	number of support vectors	1274
D_sv	dimension/features of each support vector	18

The parameters of the chosen SVM model are stated in Table 8.1. It exhibits over 99% accuracy, sensitivity and specificity. We do not examine the impact of differing arithmetic precision on how well the final SVM accelerator fares at these metrics. Although the proposed methodology is applicable in a straightforward manner in SVM implementations of differing arithmetic accuracy, we consider this decision to be a priori taken by the application designer.

4.4. Design exploration for accelerated SVM classifier

Fig. 4.3 depicts the proposed High Level Synthesis based design methodology for optimizing the architecture of SVM accelerators. The input of the flow is the C source code of the algorithmic description of SVM classifier. The methodology targets FPGA based programmable System-on-Chip platforms, i.e. Zynq [209] which provides an ARM Cortex-A9 and FPGA fabric.

All levels of accelerator optimization are performed using Vivado-HLS tool. Vivado-HLS enables user control over the synthesis process through the usage of directives. It performs some optimizations by default and also allows the user to impose directives and constraints of his own choice. The directives available from HLS aim at performance and area optimization and can be applied on functions, loops, arrays and regions containing one or more of the above. Table 4.3 summarizes the HLS directives that have been incorporated in the design exploration described in the following sections.

Regarding the proposed methodology, in the beginning the input source code is partitioned to the SVM kernel function to be accelerated and its wrapper logic which utilizes the kernel function as part of the bigger program. The SVM kernel is then optimized through the proposed two-level design optimization strategy. In parallel, the wrapper logic is tailored to the communication interface of the co-designed system.

At the first level of optimization, the SVM's source code is restructured to expose higher data- and instruction-level parallelism than the one exploited by Vivado-HLS. At the second level, the restructured code is further automatically explored over a very compact design space of memory related directives under designer and device specific constraints, e.g. maximum latency, FPGA resource utilization etc. The compact design

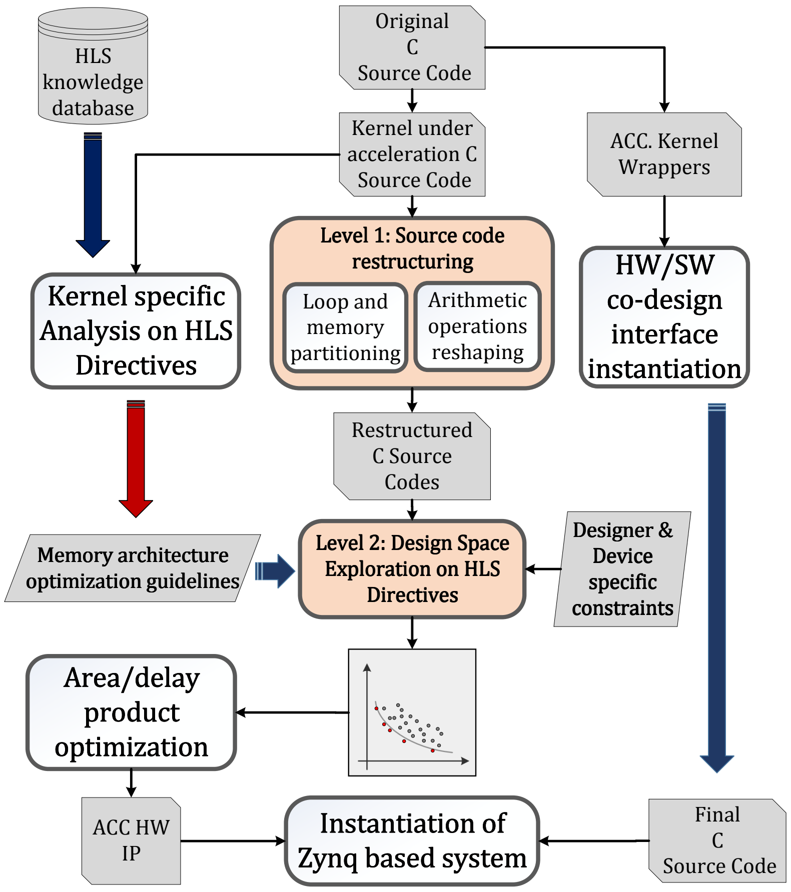

HLS knowledge database
Original C Source Code
Kernel under acceleration C Source Code
ACC. Kernel Wrappers
Kernel specific Analysis on HLS Directives
Level 1: Source code restructuring
Loop and memory partitioning
Arithmetic operations reshaping
HW/SW co-design interface instantiation
Restructured C Source Codes
Memory architecture optimization guidelines
Level 2: Design Space Exploration on HLS Directives
Designer & Device specific constraints
Area/delay product optimization
ACC HW IP
Instantiation of Zynq based system
Final C Source Code

Figure 4.3.: Proposed HLS based HW design methodology

space is defined through a set of pruning guidelines derived and validated based on extensive analysis on the impact of different HLS directives on the systems metrics and design objectives. The aforementioned analysis takes place offline and can be reused as a pre-existing knowledge database that guides the fitting of the pruning guidelines to the specific kernel instance. The proposed pruning guidelines enable optimization search to be performed in a reduced solution space enabling fast extraction of high quality design solutions.

4.4.1. Optimization Level 1: Code restructuring for HLS

We emphasize on two code restructuring strategies which if employed assist the tool to produce much more efficient HW accelerators in terms of enhanced data- and instruction-parallel accelerator implementations.

Table 4.3.: HLS directives [2]

Directive	Description
PIPELINE	Reduces the initiation interval by concurrent execution of operations within a loop or function.
DATAFLOW	Task level pipelining. Functions and loops are executed concurrently. Used to minimize interval.
INLINE	Inlines a function, removing all function hierarchy. Used to enable logic optimization across function boundaries and improve latency/interval by reducing function call overhead.
UNROLL	Unrolls for-loops to create multiple independent operations rather than serial executed ones.
ARRAY_PARTITION	Partitions large arrays into multiple smaller arrays or into individual registers to improve access to data and remove block-RAM bottlenecks.
ARRAY_MAP	Combines multiple smaller arrays into a single large array to help reduce block-RAM resources.
ARRAY_RESHAPE	Reshape an array from one with many elements to one with greater word-width. Useful for improving block-RAM accesses without using more block-RAM.

Advancing Data-Level Parallelism in HLS Through Loop and Memory Partitioning

The squared euclidean distance of the *test_vector* and each *sup_vector* is computed and used as input to the SVM kernel function. These values are weighted with a coefficient of the corresponding support vector and then accumulated in one variable to produce the classification result.

The contribution of each support vector to the total sum is irrelevant to the contribution of the others since there are no data dependencies in the computations performed between the test vector and each column of the support vector array. As a result, these computations can be performed simultaneously. This is illustrated in Fig. 4.4 where the use of different colours indicates that the computations performed between each coloured column and the test vector can be executed in parallel. Towards this goal, array *sup_vectors* can be partitioned into smaller arrays, each one containing fewer support vectors. These arrays will have the same number of rows, since the number of attributes is constant, but fewer columns. Each array will contribute a partial sum to the total sum used for classification and partial sums can be calculated in parallel. In other words, the initial problem is divided into smaller ones, which are solved concurrently. We implement data-parallel SVM kernels by first modifying the structure of the code and then utilizing HLS directives to enable parallel execution. Listing 4.2 shows the modified source code in the case that *sup_vectors* array is partitioned in two. The part of the code responsible for computing the total sum is implemented as a separate function called by the main function as many times as many array partitions exist. Each instance of this function is assigned to a different partition of the *sup_vectors* and *sv_coef* arrays and computes its partial sum. Eventually, partial sums of all partitions are aggregated by the main

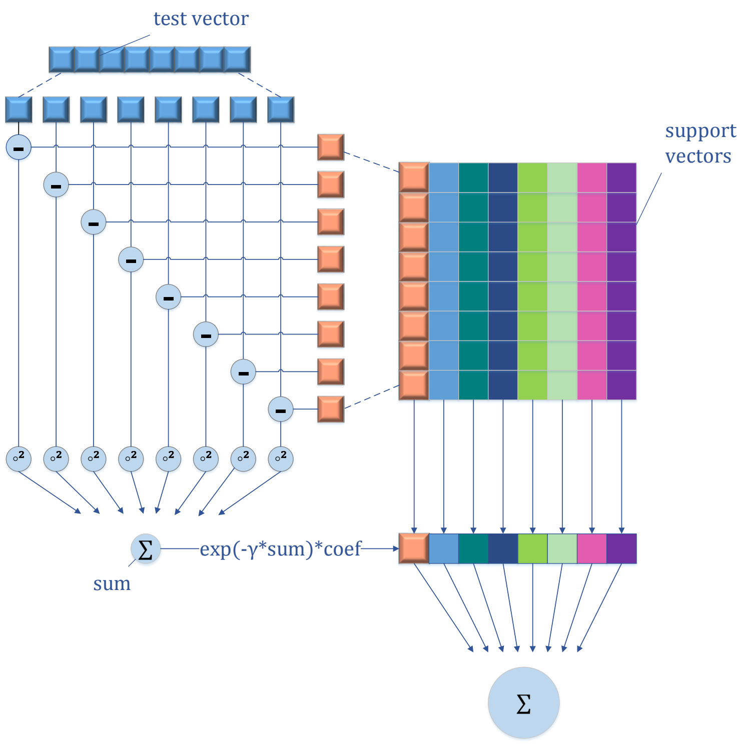


Figure 4.4.: Data-level parallelism in SVM.

function in order to produce the classification decision.

On this modified source code, we apply HLS directive *array_partition* (Table 4.3) in Vivado-HLS, for automatic array partitioning. An array in HLS is implemented using block RAMs which can have at most two read ports. In this case, simultaneous computation among all array partitions wouldn't be possible since all functions would require access to the same array at the same time. HLS would address this problem by creating a replicate for each function instance, thus leading to memory burst. By using the *array_partition* directive instead, we resolve this inefficiency since different read ports for each partition are created and parallelization is possible without any adverse effects. *Array_partition* directive is applied on *sup_vectors* and *sv_coef* arrays. Array *sv_coef* is one-dimensional, thus it is partitioned across its elements. Each partition contains consecutive elements. Array *sup_vectors* is two-dimensional and is partitioned across its column dimension into a varying number of partitions. Each one of the partitions contains consecutive columns.

In the main function we invoke the computing function once for each array partition. Its arguments include the pointers to the arrays, a value indicating the size of each partition in columns, an offset to identify the exact location of the partition in the initial array and a pointer to the address of the partial sum to be computed. Then the *dataflow* Vivado-HLS directive (Table 4.3) is applied on the main function to allow functions within its scope to operate in parallel. HLS automatically detects that function calls can be executed simultaneously and synthesis is performed accordingly.

This technique is evaluated for array partitioning of factor 2,3,4,8 and 16. The results for latency gain and resources utilization are depicted in Fig. 4.5. Latency is assessed in terms of gain, in comparison to the latency of the original code. Memory and area are presented in utilization percentages over the initial available resources.

Listing 4.2: Partitioned version of the original code

```
#define gamma 8

void SVM_partial_predict(int width,float *sum,
float test_vector[D_sv],float sv_coef[N_sv/2],
float sup_vectors[D_sv][N_sv/2]){
 int i,j;
 float diff,norma=0;
 *sum=0;

 loop_i:for (i=0; i<width; i++){
  loop_j:for (j=0; j<D_sv; j++){
   diff=test_vector[j]-sup_vectors[j][i+offset];
   norma = norma + diff*diff;
  }
  *sum = *sum + exp(-gamma*norma)*sv_coef[i+offset];
  norma=0;
 }
}

void SVM_predict(int * y){
        const float coef1[N_sv/2];
        const float coef2[N_sv/2];
        const float sv1[D_sv][N_sv/2];
        const float sv2[D_sv][N_sv/2];
        const float test_vector[D_sv];
        float diff,sum1,sum2,sum;

        SVM_partial_predict(N_sv/2,&sum1,test_vector,coef1,sv1);
        SVM_partial_predict(N_sv/2,&sum2,test_vector,coef2,sv2);
        sum = sum1 + sum2 - b;

        if (sum<0) *y = -1;
        else *y = 1;
```

}

We observe that as the number of partitions increases, the latency of the design is reduced accordingly. In fact, the speedup of execution time is very close to the ideal speed-up value, which is equal to the number of array partitions being used. For a partition factor of two, this translates into a speedup of 2×, while a partition factor of sixteen results in a speed-up close to 16×. There is a clear trade-off regarding reduced latency and increased resources utilization. For each instance of the parallel executed function the logic used for the function body is replicated. This explains the increase in DSP, Flip Flop and LUT utilization. For example, the loop body requires 45 DSP units for all the operations to be scheduled the way they are in the original code. This number remains the same for each independent instance but the total number is multiplied by a factor equal to the number of instances or partitions used. This explains why values over 100% utilization are presented for a large partition factor, meaning that resource availability of the target platform has been exceeded. The same principle applies for the utilization of Flip Flops and LUTs.

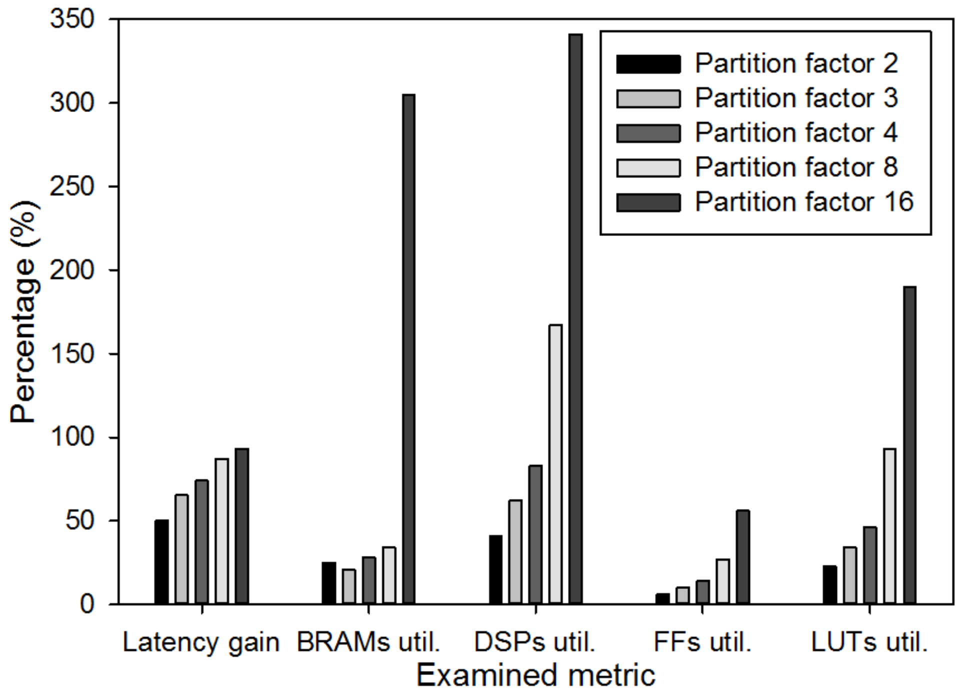


Figure 4.5.: Performance and utilization for increasing number of partitions (automatic)

An interesting parameter is the utilization of BRAM for array implementation. By using the partitioning directive we have managed to avoid the replication of the arrays for each instance. Each function requires as many BRAMs as are needed to implement its partition of the array. This leads to an almost constant total number of used BRAMs in all cases except for the one of partition factor 16. In that case HLS demonstrates an inconsistency and replicates of the arrays are created, resulting in a burst in memory usage.

In order to avoid this memory burst, manual array partitioning is applied by allocating several, smaller arrays from the beginning instead of declaring one large array and then partitioning it into smaller ones using the corresponding directive. Now in each function call, a different array pointer is passed pointing to one of the array partitions. This means that the programmer must manually split the array containing the support vectors and the array of their coefficients into smaller arrays containing fewer support vectors. As expected, the increase in area resources (DSPs,Flip Flops and LUTs) remains the same, in both cases. However, the burst in number of BRAMs when partitioning for a factor of 16 is now eliminated and BRAM utilization remains practically the same regardless of the number of partitions. The gain in latency of automatic vs manual array partitioning is presented in Fig. 4.6. *We observe that the proposed manual partitioning is more effective than the automatic one, giving a speed-up practically equal to the ideal one.*

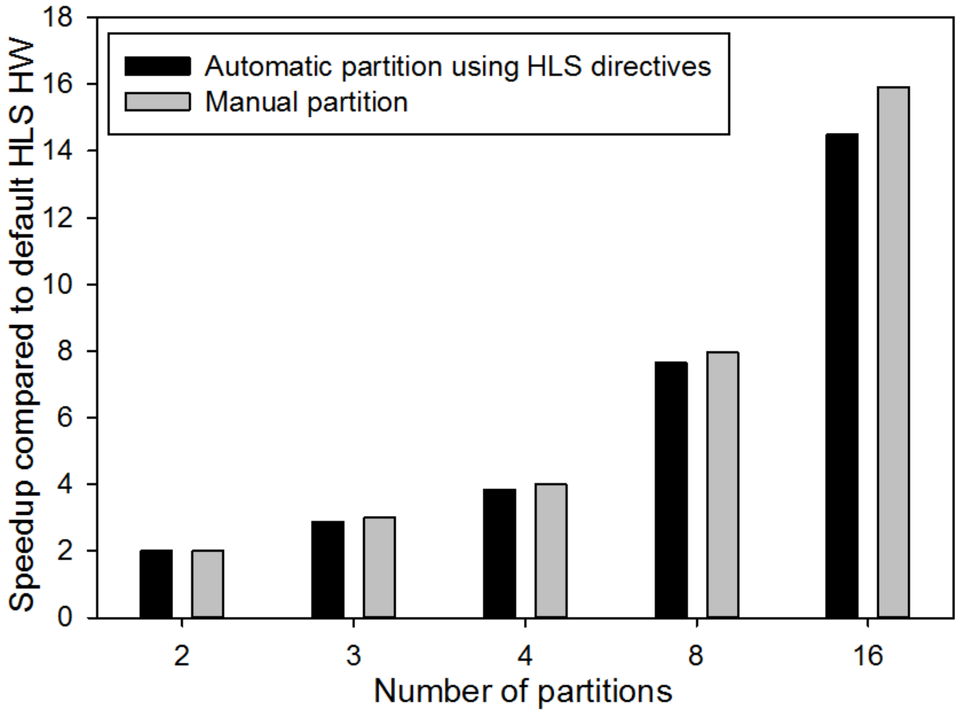


Figure 4.6.: Speedup gain comparison (automatic vs manual)

Advancing Instruction Level Parallelism through arithmetic operation reshaping

While data-parallel optimization enables different support vectors to be calculated simultaneously, instruction-level parallelism targets at optimizing the performance of computations required for calculating the contribution of one support vector to the classification result. This contribution requires the computation of the squared euclidean distance between the current support vector and the test vector, performed in the inner loop, i.e. *loop_j*. The loop iterates over the elements of the two vectors, computes their difference, multiplies it with itself and accumulates this squared difference to a variable that holds the squared euclidean distance when the iterations end. Applying the loop unroll directive (Table 4.3) to this loop should increase ILP since the computations regarding each element of the vectors can be performed independently of each other and thus in parallel.

Automatic loop unrolling of the inner loop using the HLS loop unroll directive leads to a reduction of the accelerator latency, as it can be seen in Table 4.4. The result however is not the one anticipated. *Careful inspection of the scheduling reports generated by the tool showed that it does not fully exploit the available parallelism.* The subtractions and multiplications of different elements are not scheduled simultaneously and additions are scheduled in a serial manner even though there is no true data dependency between the added values.

Inspired by several works in computer arithmetic [210–212], we propose a more efficient implementation of this loop unrolling by modifying the structure of the addition and transforming it into a tree-based computation of the final result. In this way, HLS can schedule the independent calculations in parallel by allocating more hardware resources, if needed. The structure of the tree based data calculations is depicted in Fig. A.8 for an unroll factor of value 6. The number of loop iterations is reduced to 3. In the internals of the third iteration, it is shown how *input_vector* and *support_vector* are subtracted, squared and finally added in different levels of a tree-like structure. Operations in each level can be executed in parallel. To produce the final result of a full *loop_j* execution, the outcome values of each iteration have to be accumulated in one variable. This is also performed in a tree-like way and thus it does not introduce any more latency to the hardware. In the proposed framework, tree-base code restructuring is automatically performed through a custom source-to-source transformation script that generates the tree reconstructed SVM source code according to the level of the loop unrolling factor.

The proposed manual unrolling technique was examined for unrolling the inner loop of the SVM code 3, 6 and 18 times, respectively. Table 4.4 reports the difference in performance and resources utilization when applying the unroll directive on the inner loop versus manually unrolling it while using a tree-based expression balancing structure for the computations. *Significant improvement in latency gain is observed when*

Table 4.4.: Evaluated metrics for automatic vs manual unrolling

Version	Unroll factor	Automatic Unroll using directives					Manual Unroll using tree structure				
		latency (cycles)	BRAM (%)	DSP (%)	FF (%)	LUT (%)	latency (cycles)	BRAM (%)	DSP (%)	FF (%)	LUT (%)
initial	-	412783	24	20	3	11	412783	24	20	3	11
unrolled	3	252259	24	21	3	11	214039	47	26	4	14
unrolled	6	206395	24	23	3	11	149065	70	34	5	18
unrolled	18	173271	27	20	3	11	90461	27	50	8	29

applying manual instead of automatic unrolling. The greater the unroll factor is, the higher the performance gap of the two methods is. The reason is that the tree based structure of performing computations allows data independent operations to be executed in parallel, thus significantly reducing inner loop latency and subsequently total design latency.

The expected trade-off of the proposed technique is increase in resources utilization and specifically in DSPs, FFs and LUTs. The implicit declaration of parallel subtractions and multiplications as well as of the parallel additions of the tree structure results to an increase in the number of computational instances required to achieve instruction level parallelism. In other words, the HLS tool identifies the available parallelism and allocates more logic to exploit it.

Listing 4.3: Unrolled version of the original code
```
#define gamma 8

const float sv_coef[N_sv];
const float sup_vectors[D_sv][N_sv];

void SVM_predict (int *y,
        float test_vector[D_sv]){

        loop_i:for (i=0; i<N_sv; i++) {
                loop_j:for (j=0; j<D_sv; j=j+6) {
                        d1=test_vector[j]-sup_vectors[j][i];
                        d2=test_vector[j+1]-sup_vectors[j+1][i];
                        d3=test_vector[j+2]-sup_vectors[j+2][i];
                        d4=test_vector[j+3]-sup_vectors[j+3][i];
                        d5=test_vector[j+4]-sup_vectors[j+4][i];
                        d6=test_vector[j+5]-sup_vectors[j+5][i];

                        sq_prod1=d1*d1;
                        sq_prod2=d2*d2;
                        sq_prod3=d3*d3;
```

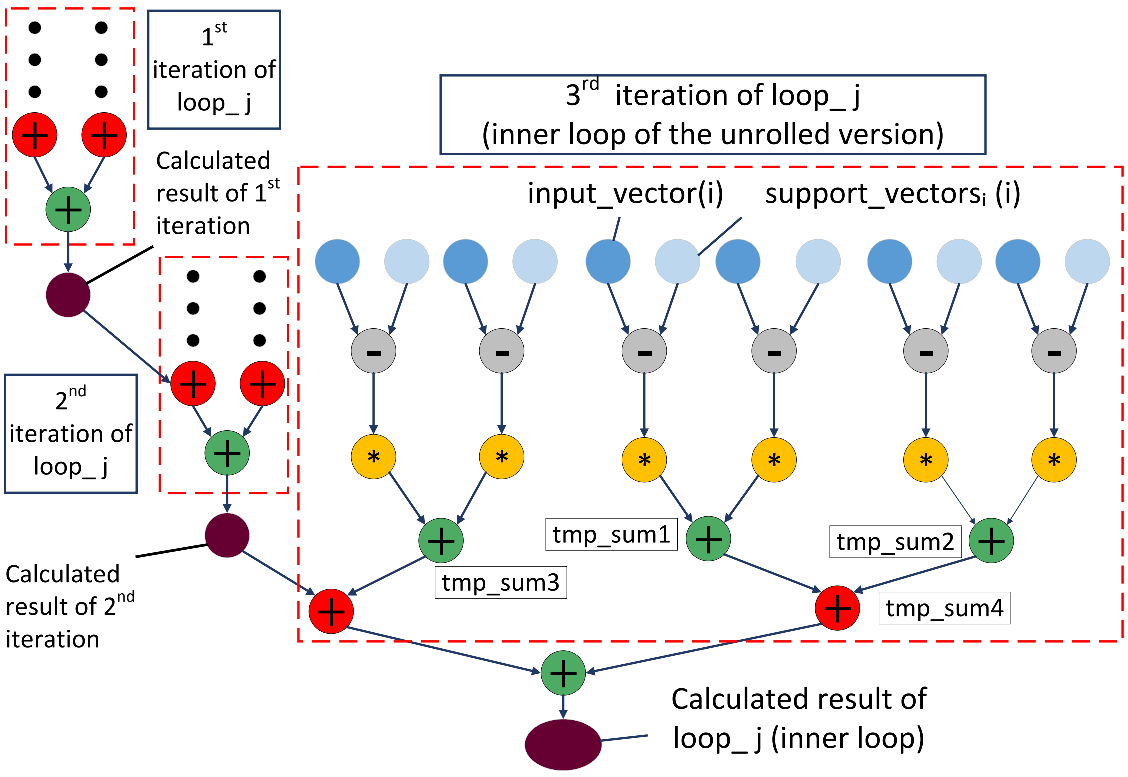

1st iteration of loop_ j
Calculated result of 1st iteration
3rd iteration of loop_ j (inner loop of the unrolled version)
input_vector(i)
support_vectorsi (i)
2nd iteration of loop_ j
tmp_sum1
tmp_sum2
tmp_sum3
tmp_sum4
Calculated result of 2nd iteration
Calculated result of loop_ j (inner loop)

Figure 4.7.: Tree based computations for manual unrolling and HLS scheduling

```
                    sq_prod4=d4*d4;
                    sq_prod5=d5*d5;
                    sq_prod6=d6*d6;

                    tmp_sum1=sq_prod1+sq_prod2;
                    tmp_sum2=sq_prod3+sq_prod4;
                    tmp_sum3=sq_prod5+sq_prod6;

                    tmp_sum4=tmp_sum1+tmp_sum2;
                    norma = norma + tmp_sum3;
                    norma = norma + tmp_sum4;
          }

          sum = sum + exp(-gamma*norma)*sv_coef[i];
          norma=0;
}

sum = sum - b;
if (sum<0) *y = -1;
```

```
        else *y = 1;
}
```

An adverse and unexpected effect of manual loop unrolling regards BRAM utilization. For an unroll factor of 3, three accesses to support vectors array are required simultaneously but the BRAM has at most two read ports. To achieve the reading of three elements, HLS creates a copy of the array and implements both copies using dual port RAMs to provide four reading ports. As a result, there is a gain in parallelism and latency but BRAM utilization doubles, i.e. 128 allocated BRAMs instead of 64. In the case of unroll factor of 6, three copies of the same array implemented with dual port BRAMs are required thus BRAM utilization triples. Similarly, when fully unrolling the loop we would expect nine copies of the array to be created in order for 18 elements of the same array to be read concurrently. However, HLS automatically partitions the array in 18 rows, each one being implemented as dual port BRAM, thus allowing simultaneous access without memory increase. Consequently, HLS behaves in an inefficient and inconsistent manner. To tackle this issue and generate efficient IPs both in terms of performance and resources utilization, array partition directives can be applied to the arrays which is fully examined in section 4.4.2.

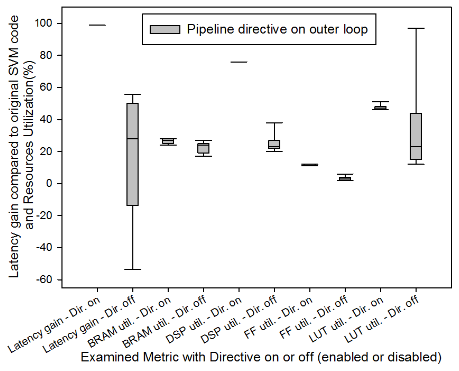
4.4.2. Optimization Level 2: Design Space Exploration of HLS Directives

In this section, we focus on the exploration of HLS knobs exposed as Vivado-HLS directives to enable further SVM accelerator tuning. Analysis of the impact of directives is performed using Vivado which is an industrial strength HLS tool and also automates the instantiation process of a Zynq based co-designed system. However, any other HLS tool which uses annotation of the input source code like LegUp [213] could be incorporated in our proposed framework and derive an analysis similar to the one presented in following sections.

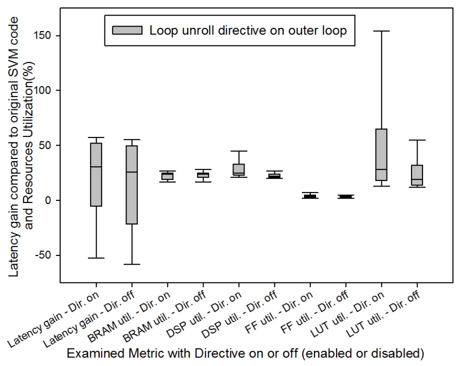
Analysing impact of directives in SVM classifier

In previous sections, techniques based on structural modifications of the C code were developed to create effective RTL architectures. These formed a first level of optimization that brings to RTL the required efficiency. The performance though can be further improved by meticulously tuning the available HLS directives. In this section we are going to explore the impact of the chosen directives on the efficiency of the accelerator.

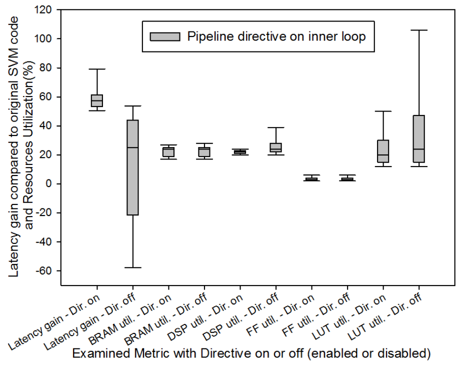
The directives applied on the classifier are **Loop pipeline**, **Loop unroll**, **Array partition** and **Array reshape** (Table 4.3) [145]. A short description of these directives is



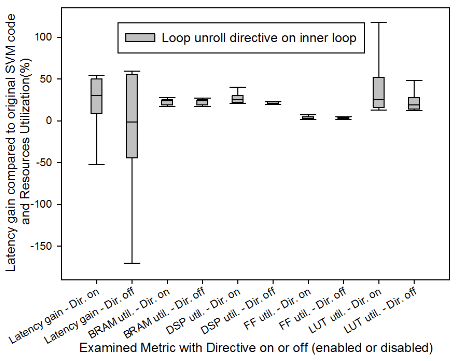
(a) Impact of pipeline directive in loop_i



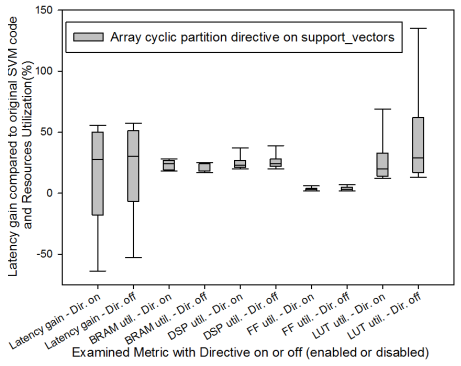
(b) Impact of loop unroll directive in loop_i



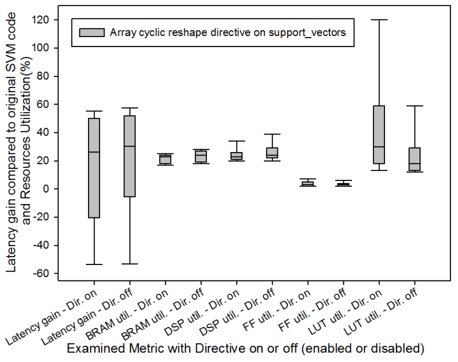
(c) Impact of pipeline directive in loop_j



(d) Impact of loop unroll directive in loop_j



(e) Impact of array partition cyclic directive in sup_vectors



(f) Impact of array reshape cyclic directive in sup_vectors

Figure 4.8.: Impact of directives on SVM kernel source code

Table 4.5.: Applied directives and their parameters

directive	variable	factor	dimension
pipeline	loop_i	on,off	-
	loop_j	on,off	-
loop unroll	loop_i	0,2,7	-
	loop_j	0,2,3,4,6,9,18	-
partition	sup_vectors	0,2,3,4,6,9,18	rows (dim 1)
	test_vector	0,2,3,4,6,9,18	-
	sv_coef	0,2,7	-
reshape	sup_vectors	0,2,3,4,6,9,18	rows (dim 1)
	test_vector	0,2,3,4,6,9,18	-
	sv_coef	0,2,7	-

available on Table 4.3. They were selected with the intent of increasing the parallelism in the computation of the squared euclidean norma in the inner loop. The loop is unrolled and directives *partition* and *reshape* are applied to the accessed arrays, to increase the number of their ports or their word-width respectively and allow simultaneous access to their elements. In the same manner, these directives can also be applied to the outer loop and the arrays referenced by its iterator.

Table 4.5 includes the directives applied to each variable in the SVM code in Listing 4.1. In the column named "factor", the different possible values for each directive are presented. An exploration has been performed to analyse the impact of these directives on the SVM classifier code. The full search space consists of all valid combinations of these directives for all values of their parameters included in Table 4.5.

This would lead to an immense search space, that can however be reduced. The configurations including partitioning and/or reshaping array sv_coef are excluded since initial profiling did not indicate paramount effects. Furthermore, some of the combinations are excluded from the search space because they lead to equivalent design configurations or contain incompatible directives. More specifically:

- **Pipeline directive causes automatic unrolling of all loops down its hierarchy.** This behaviour leads to two constraints that significantly reduce the exploration space. When the outer loop is pipelined, there is no point in unrolling the inner loop by any factor less than the one corresponding to a fully unrolled loop. When the outer loop is pipelined, there is no point in pipelining the inner loop since it no longer exists as one.

- **Fully unrolling the inner loop leads to automatic fully partitioning of the support vector into separate rows and of the test vector into registers.** When the inner loop is fully unrolled, there is no point in partitioning or reshaping the arrays referenced inside the loop by a factor less than the one corresponding to fully partitioning or fully reshaping.

The result is a full search space consisting of 86580 configurations. Some of them cannot be synthesized while others are synthesized but exceed time constraints. This leads to a full space of 70962 configurations. For each one of them latency and area utilization information are available after synthesis results are acquired.

Box-plots of Figures 4.8a to 4.8f summarize the results of exploration regarding the impact of each directive on latency gain, BRAM, DSP ,FF and LUT utilization. Each diagram in Fig. 4.8 corresponds to an assessment of the effects of a single directive. For each metric, the left box-plot groups together all configurations that include the directive under assessment. The right box-plot includes the rest of the configurations. Outliers have not been included.

In Fig. 4.8a applying pipeline directive on the outer loop (loop_i) is examined. Pipelining the outer loop leads to unrolling all loops down its hierarchy and therefore the inner loop. When this directive is applied, latency gain reaches 99%. Its side-effect is slightly increased BRAM utilization due to the automatic unrolling of the inner loop, which in turn causes the automatic partitioning of the arrays referenced inside it. DSP, Flip Flop and LUT utilization change significantly, exhibiting greater median and range values. This is attributed to the automatic unrolling of the inner loop which replicates logic.

Fig. 4.8b depicts the results of outer loop unrolling (loop_i). This does not seem to be crucial in the improvement of the design in terms of latency since the range of values extends from negative values to 65% in both cases. BRAM utilization is not greatly affected by the directive. The median value is the same in both cases and equal to the original one. As far as DSPs, FFs and LUTs utilization is concerned their value range is wider and their median value is greater when the directive is applied. This is again anticipated, since unrolling a loop in HLS duplicates the loop body and thus the utilized area is increased.

When applying the pipeline directive to the inner loop (loop_j), latency gain ranges between higher values and has a median value of 60% as shown in Fig. 4.8c. FPGA resources utilization exhibits narrower range with smaller median values. Parallelism is increased by using all resources and not by replicating hardware. Operations are scheduled when the required resources and data are available and not necessarily sequentially. This increases instruction level parallelism and reduces the initiation interval by allowing the concurrent execution of operations within the loop leading to improvement in latency gain.

Unrolling the inner loop has a positive impact on latency. It limits considerably the range of values of latency gain and the median value is greater. This is illustrated in

Fig. 4.8d. BRAM utilization remains constant and equal to the initial utilization when no directives are applied. Additionally, when the inner loop is unrolled DSP, FF and LUT utilization range is wider and in a higher region. This is again due to the fact that unrolling a loop leads to replicating the loop body and thus to more logic and area.

Partitioning an array aims at implementing each partition in different BRAMs thus creating more read ports and allowing higher parallelism. As shown in Fig. 4.8e, this higher parallelism results indeed in increased BRAM utilization, when partitioning of *sup_vectors* array is applied. The directive does not have great effect on latency or DSP and FF utilization. LUT utilization seems significantly greater when the partitioning is not applied but this cannot be fully justified due to the noise in measurements inserted by other directives. Similar conclusions can be drawn for the application of the reshape directive presented in Fig.4.8f. BRAM utilization in that case is worth mentioning as it decreases both in range and median value since the reshape directive re-combines the elements created by partitioning into a single block-RAM with wider data ports.

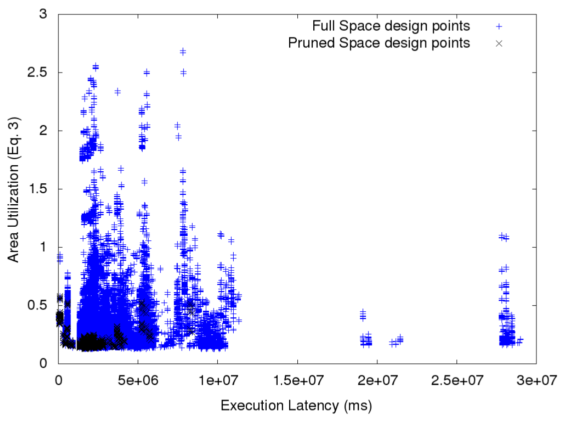
HLS directives design space pruning guidelines

Although an exhaustive search of the directives related exploration space guarantees the discovery of optimal configurations, it requires extremely long run-times, thus forming an impractical solution. A more targeted space exploration disposed of suboptimal design configurations, is highly desired in designing complex accelerator architectures in order to locate optimal configurations in less time.

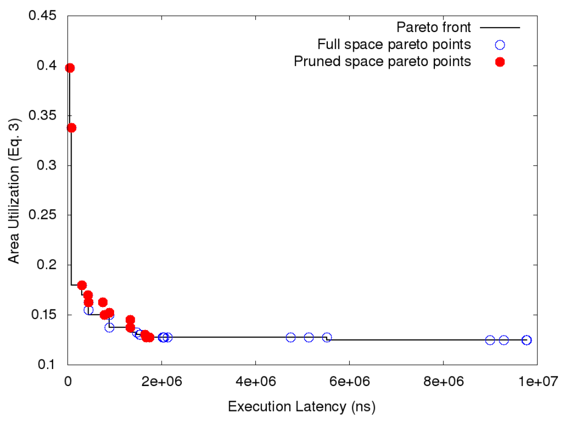
In this section, we define and analyse three pruning guidelines which by effectively parallelizing the inner loop of the algorithm (loop_j) succeed to exclude suboptimal design points of high execution latency from the search space. *The proposed guidelines are based on the observations of HLS directives impact analysis provided in Section 4.4.2.* In next section, we quantitatively validate their effectiveness in respect to their optimization potential. The design space pruning guidelines follow the principle of customizing the accelerator's memory architecture to its computation and memory access patterns. The only prerequisite for applying them is that the elements which need to be accessed concurrently each time, maintain their offset in the initial array. They are summarized as follows:

PG1. *When the inner loop is unrolled by a factor, if the arrays referenced by the loop iterator are partitioned, the partition factor should be equal to the unroll factor.*

PG2. *When the inner loop is unrolled by a factor, if the arrays referenced by the loop iterator are reshaped, the reshape factor should be equal to the unroll factor.*



(a) Combined Full and Pruned search space



(b) Pareto points of Full and Pruned search space

Figure 4.9.: Full and Pruned Design Space

PG3. *When the inner loop is unrolled by a factor, if the arrays referenced by the loop iterator are both partitioned and reshaped, the product of partition and reshape factor should be equal to the unroll factor.*

The above rules address two main issues that arise when unrolling the inner loop. First, unrolling a loop implies that multiple elements of arrays referenced inside it, need to be manipulated in parallel. Each array is implemented as BRAM with two read ports at most, thus a memory bottleneck is observed that limits potential performance gain despite the large parallelism potential. Array partitioning leads to memory partitioning into several banks thus allowing simultaneous access to multiple elements of the same array. Array reshaping recombines the elements of array partitions into a single BRAM that has wider data ports, allowing access to multiple elements of the initial array with a single read. The combination of the two directives also allows simultaneous access to multiple elements. In total, to fully exploit the inherent parallelism, array restructuring must allow simultaneous access to at least as many elements as are referenced by the loop iterator, i.e. equal to the unroll factor.

On the other hand, partitioning or reshaping an array by a factor greater than required can have adverse effects. Since automatic partitioning and reshaping does not change the source code of the accelerator, array references are still tuned for the initial array. As a consequence the HLS tool compiler adds modulo operations to determine to which array partition the elements referenced in each iteration belong and a significant overhead to the total accelerator latency is introduced. The greater the partition factor is, the greater the overhead gets. Similarly, when reshaping an array, packing more elements than required in a single element of wider word width, introduces overhead to segment the part of the word that is actually required. Therefore, packing all the elements required with no inclusion of redundant elements is more efficient.

Validation of the proposed pruning guidelines

The intuitive guidelines of the previous section need to be validated in a more robust way. Therefore, it is necessary to study the set of all possible configurations created by combining inner loop unrolling with partitioning and/or reshaping the arrays referenced inside the loop. No other directives are included in the exploration, to ensure that results are not distorted.

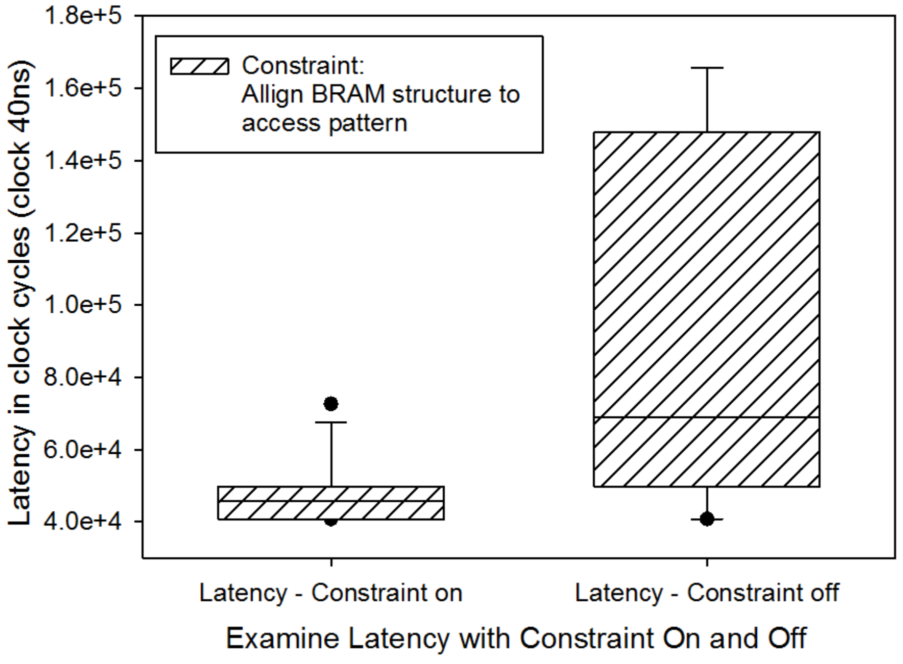


Figure 4.10.: Impact of loop unroll directive in loop_j

The guidelines are applied on this set. This creates a pruned set that only contains the configurations of minimum latency. The left box-plot of Fig. 4.10 (Latency - Constraint on) contains these configurations while the right box-plot (Latency - Constraint off) the discarded ones. The effectiveness of the pruning guidelines is proven by observing that the vast majority of low latency design points are included in the pruned set.

The pruning guidelines can now be applied on the full search space. The resulting pruned space should be evaluated in terms of its efficiency to locate the optimal points. For that purpose, a Pareto analysis is performed on both spaces. This Pareto analysis considers the trade-off between delay and area utilization. The area utilization metric combines BRAM, DSP, Flip Flop and LUT utilization percentage as shown in 4.3[1].

[1]The values are provided as percentages in HLS reports. To get their average value, their sum is divided by 400 instead of 4

$$Area_{util} = \frac{BRAM_{util} + DSP_{util} + FF_{util} + LUT_{util}}{400} \tag{4.3}$$

Fig. 4.9a depicts the mapping of the pruned space (black 'x' symbols) on the full space (blue '+' symbols) and Fig. 4.9b includes the Pareto points of the two design spaces. Although full search space exploration is very time consuming[2], it was performed to enable validation of the efficiency of our approach. We do not consider that this full search is practical under realistic design time requirements especially when numerous SVM models need to be optimized.

Statistical data are necessary for a fair comparison between full and pruned space. The full design space for the SVM classifier includes 70962 configurations from which 30 distinct Pareto points are identified (Fig. 4.9a). *The pruned space is composed of merely 2212 configurations, i.e. around 3.1% of the initial solution space including 13 Pareto points. 10 of them are included in the pareto points of the full space, resulting in Pareto coverage of 33%. Therefore the proposed pruning guidelines deliver an extremely reduced design space spreading across the delay-area optimality region of SVM accelerator design solutions.*

Delay-Area Product Optimization

As shown in the previous section, the proposed pruning guidelines result in a significant reduction, around 97.44%, of the original solution space that concentrates on the Pareto optimal region considering the delay-area metrics (Fig.4.9). Although feasible, an exhaustive evaluation based on exhaustive search optimization of the pruned solution space remains quite impractical, requiring around 5 hours of execution time. In addition, the derivation of a Pareto-front as the main output of the exploration procedure enables a more deep analysis on the delay-area trade-offs, however it leaves to the designer the final decision on which SVM design configuration should be selected and implemented, i.e. the more the Pareto points the more difficult the decision process.

In order to enable a faster and thus more practical optimization procedure than full search as well as to provide to the designer a single optimized design solution, we implemented a single-objective optimization framework, as the final stage of our methodology, that receives as input the pruned design space, D, and solves the following optimization problem:

$$\min_{x \in \boldsymbol{D}} \left[\; Delay(\mathbf{x}) \times Area_{util}(\mathbf{x}) \; \right] \in R^2 \tag{4.4}$$

[2]Approximately 15 days on an Intel Xeon CPU E5-2650 v2@2.6 GHz, 64 GB RAM

subject to:

$$\begin{bmatrix} BRAM_{util}(\mathbf{x}) \\ DSP_{util}(\mathbf{x}) \\ LUT_{util}(\mathbf{x}) \\ FF_{util}(\mathbf{x}) \end{bmatrix} \leq \begin{bmatrix} 100\% \\ 100\% \\ 100\% \\ 100\% \end{bmatrix} \tag{4.5}$$

The optimization goal is to find the configuration vector, $\mathbf{x}$, that minimizes the delay-area product in a solution space. A new possible design point is evaluated in terms of the optimization objective and then filtered according to all inequalities of Eq. 4.5. Given the compact solution space, we employ discrete steepest decent greedy optimizer to solve the aforementioned optimization problem. Given an initial set of randomly selected points from the pruned design space, it starts to greedily move within the design space towards a local minimum following the negative of the gradient.

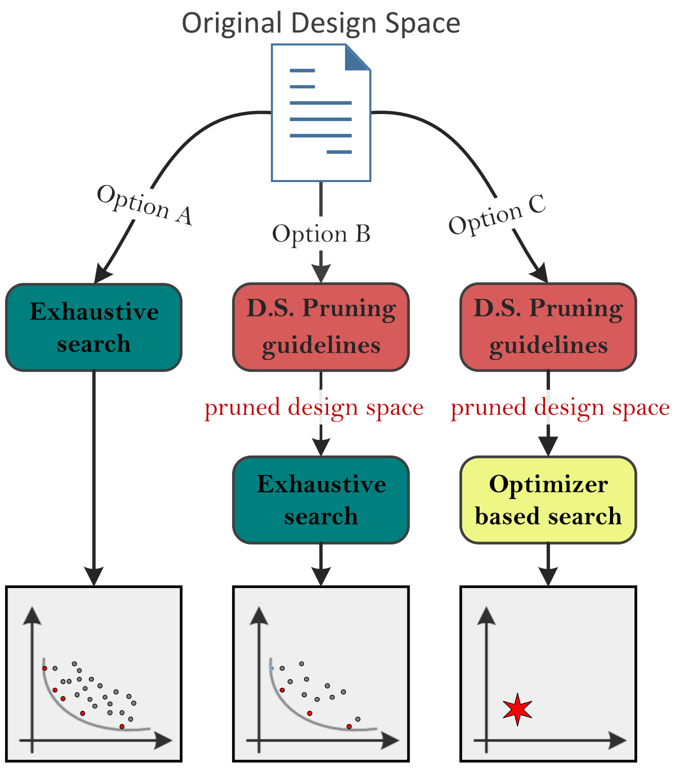


Figure 4.11.: DSE options provided by the proposed framework

In total, Fig. 4.11 illustrates and summarizes the three different options provided to a designer utilizing the presented framework given an input design space which consists of a source code description of the function to be accelerated and a set of HLS directives which can be applied on this code. In *Option A*, the framework is instructed to perform an exhaustive exploration of all the design space. It is the most time consuming option (15 days for the case study) and its outcome is considered optimal since the input design space is fully explored. In *Option B*, the previously described design space pruning guidelines are enforced on the input design space to produce a reduced design space which is then exhaustively explored. It is a medium case regarding time consumption (5 hours for the case study) and is able to produce a sufficient number of optimal design points. Last but not least, in *Option C* the input design space undergoes pruning but instead of an exhaustive traversal of the pruned space, an optimizer is employed providing the designer a single design point within a limited time frame (a few minutes for the case study), which optimizes a single objective.

4.5. Design Methodology for Approximate SVM

In this section we describe the methodology followed to create an efficient approximate SVM kernel that performs arrythmia detection on ECG signals. First, we present the exploited approximation techniques and then we apply the proposed framework of Section 4.4.2. Considering the data dependencies of the kernel and the complexity of the performed computations, we apply approximate techniques, that reduce both the number and complexity of the performed operations, as well as high level synthesis optimization techniques that boost performance. For the rest of the section, the accuracy of the approximate SVMs refers to the percentage of correct classifications in a set of heart beats.

4.5.1. Approximate Techniques

Loop Perforation

The first approximate technique that we examine is loop perforation [214], i.e., omitting loop iterations. Loop perforation was introduced in software approximation, but since it is an algorithmic technique it can be equivalently applied in HLS. Although it can be applied to any algorithm, its optimal tuning is application-specific. In this case, the data dependencies of the SVM accommodate loop perforation. As seen in Listing 4.1, the squared euclidean distance of a single *test_vector* and each *sup_vector* is computed (*norma*), filtered through the RBF kenrel and eventually accumulated to a variable that defines the classification result. The contribution of each support vector to the total sum is irrelevant to the contribution of the others. Hence, loop perforation appears to be a very promising approximation candidate for SVM that allows us to eliminate operations (and thus the total latency of the kernel), as well as the area footprint, at a small accuracy loss.

In order to efficiently apply loop perforation, we utilize a greedy approach to identify the

Table 4.6.: Impact of Loop Perforation on SVM Accuracy.

Perforation %	Accuracy%	Speedup
2%	98.02%	1.41
4%	97.12%	1.45
6%	96.27%	1.47
8%	92.85%	1.51
10%	90.43%	1.54

support vectors to be omitted. In this study, we consider loop perforation ranging from 2% up to 10% with step 2%. Given a target of p% perforation, our greedy algorithm implements an iterative procedure. In each iteration, we perforate an additional support vector. The iterations terminate when p% perforation is reached. Given the support vector matrix of the previous iteration, we evaluate the classification accuracy when also perforating each one of the remaining support vectors. Then we perforate the respective vector that achieves the highest accuracy. Table 4.6, presents the accuracy and speedup over the exact-SVM, of the SVMs produced by our greedy approach. The attained accuracy ranges from 98.0% for 2% perforation target to 90.4% for 10% perforation target. The respective speedups range from 1.41× to 1.54×. As it can be seen, in the latter case extra perforation comes at a significant accuracy loss without compensating with significant additional speedup. Further perforation is not of interest in this application due to lower accuracy rates.

Precision Scaling

Vivado HLS provides fixed-point precision data types for C/C++ kernels. We can leverage this feature to apply precision scaling, i.e., implement the SVM kernel with smaller bit-widths. This will result in smaller hardware operators and thus faster circuit.

Migrating from standard C types to arbitrary precision types is not trivial in terms of maintaining the correctness. For that reason, we perform an exploration to refine the utilized data types to their optimal size. Six different data types are defined for variables $test\_vector$, $sup\_vectors$, sv_coef, $diff$, $norma$, sum of Listing 4.1. We use fixed point representation for each data type and examine varying precision for the decimal part that ranges from 12 up to 22 bits. In our exploration we evaluate all the possible combinations with respect to the data type and its precision. For example Table 4.7 shows the precision assigned to each data type for the most accurate configuration resulted by this exploration. The accuracy of this configuration is 99.82% and its attained speedup is 2× with respect to the exact implementation.

Table 4.7.: Fixed point Data types Initial Configuration in Bits

Variable	Bit-Width	Integer Bits	Decimal Bits
$test\_vector$	24	2	22
$sup\_vectors$	24	2	22
sv_coef	32	10	22
$diff$	25	3	22
$norma$	31	9	22
sum	32	10	22

4.5.2. Approximation and Optimization Methodology

The techniques described so far should be combined in a synergetic and strategic manner to deliver a highly optimized approximate SVM classifier. The non-linear and nontrivial inference effects between the applied approximation techniques as well as the HLS optimization directives regarding both performance as well as accuracy require the use of heuristic algorithms in practice. A brute-force evaluation of all possible combinations of the examined techniques, would lead to an enormous design space. In order to avoid an exponentially growing search space, we propose a framework that incrementally exploits each technique and gradually builds up to the implementation of an efficient FPGA SVM classifier, accelerated through both approximate and high level synthesis optimization techniques.

As a first step, we apply loop perforation, carefully adjusted to the kernel's requirements. Specifically, loop perforation technique is applied to the exact SVM classifier, using the greedy algorithm in Section 4.4.2. This step results in constructing five approximate SVM classifiers, each one exhibiting a perforation percentage of 2%,4%,6%,8% and 10% respectively and an initial boost in performance.

As a second level of optimization, we explore the performance enhancement that can be extracted by meticulously tuning the in-build optimization knobs of HLS tool with respect to the previously conducted approximations. To avoid an exhaustive design space exploration, we utilize the framework proposed in [178], which elegantly prunes the design space and efficiently converges towards the Pareto front. The basis of the proposed pruning strategies lies in customizing the memory architecture according to the computation and memory access patterns of the algorithm. Since different perforation percentages result in different memory layouts and thus different optimal configurations of HLS directives, each approximate-perforated SVM should be evaluated separately. Therefore, we perform an efficient design space exploration and acquire a resource utilization-speedup Pareto-front for each approximate SVM resulted by loop perforation. Note that all design points within the same Pareto exhibit the same classification accuracy.

On the last optimization level, we further boost the performance by applying precision scaling. For each Pareto point, an exploration is performed to refine the utilized data types to their optimal precision. We evaluate all the possible combinations (Section 4.5.1), and a new Pareto Front is extracted. The approximate designs in this Pareto front, apply both loop perforation and precision scaling, as well as HLS optimization.

4.6. Experimental Results

In this section, the efficiency of the proposed design methodology is experimentally evaluated. We provide three different evaluation analyses regarding i) the efficiency of the proposed design exploration to deliver delay-area optimized designs compared typical exploration approaches, ii) the efficiency and robust behavior of the derived SVM accelerator design configurations at scale, i.e. for SVM configurations exposing increased computational complexities, iii) the effectiveness of the proposed approach for co-designed ECG analysis flows and iv) applying the proposed framework to an approximate SVM kernel.

4.6.1. Experimental Set-up

The primary target FPGA board in this work is Zedboard Zynq Development Kit xc7z020clg484-1 [208]. It provides a complete ARM based Processing System (PS) featuring a Dual ARM Cortex-A9 MPCore with integrated memory controllers, floating point operations support and full Linux OS compatibility. The PS side of the board is tightly integrated with the Programmable Logic (PL). The FPGA resources provided by the target board are adequate to support all proposed HW optimizations on the utilized SVM model. This model itself is constrained in its parameters size due to the fact that it has been produced via a structure/accuracy optimization search process [207]. In general, available FPGA resources are a major constraint of the HW optimization process and this has also been reflected in Section 4.4.2. However, providing that resources are available, the proposed framework can maintain efficiency of the derived HW accelerators when SVM model parameters are scaled (Section 4.6.3).

Xilinx Vivado-HLS (v2015.2) [145] has been used to derive all SVM accelerators mentioned in the experimental evaluation. The same tool was also utilized to instantiate the HW/SW co-designed ECG analysis system on the target board. To achieve communication of the PS system with the HW accelerators implemented on PL side, ARM AXI interfaces are used. AXI is part of Advanced Microcontroller Bus Architecture (AMBA). There are different types of AXI interfaces available for the target Zynq board. In this work AXI4 Lite interface is utilized, which is a subset of AXI4 Memory Mapped Interfaces. The Processing System implements the Master Interface of the AXI4 bus while the IP the Slave

Interface, which is controlled by the Master through block level signals. The AXI4 Slave Lite Interface is added to the IP that will configure the PL.

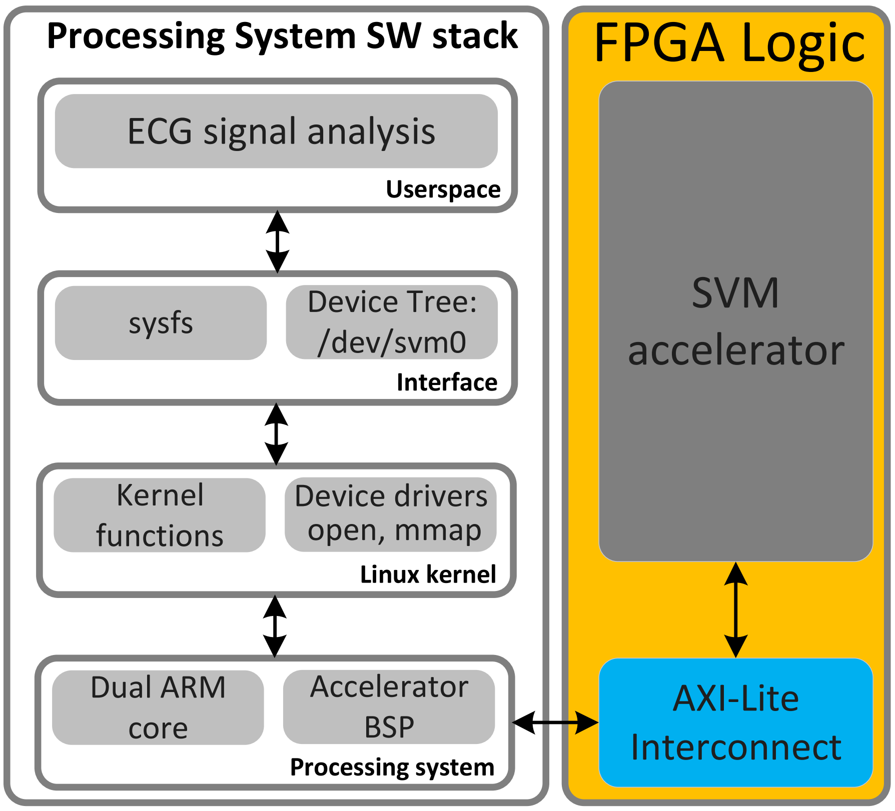


Figure 4.12.: Target Zynq based system HW/SW overview

The overview of the HW and SW parts of the co-designed system is illustrated in Fig. 4.12. The software is developed in Xilinx SDK 2014.4 and is executed on the ARM Cores of the PS side. The ARM cores run Linux OS developed using Petalinux 2014.4 tool. Petalinux uses hardware configuration files generated by Vivado to build a Linux distribution that includes all drivers necessary for the communication between the PS and PL, including the custom IP. The IP is included in the Linux file system as a userspace I/O device and is mapped to memory via mmap system call. The software application uses this device as a common file and accesses it through a pointer to the corresponding mapped memory range. The accelerator Board Support Package provided by Vivado, encloses the information of the memory mapping of the registers of the IP to the ARM CPU. Using this mapping, the CPU feeds the accelerator IP with data which in turn returns the classification result to the CPU.

4.6.2. Efficiency evaluation of the proposed DSE methodology

In this section, we evaluate the efficiency of the proposed pruning based design exploration strategy for SVM accelerator optimization in comparison to exhaustively traversing the original design space.

To quantify the efficiency of the proposed exploration strategy, we make use of two optimization meta-heuristics to search for design points inside the full and the pruned design space. These two optimization meta-heuristics are i) steepest descent (Section 4.4.2) and ii) a genetic optimizer[3]. All exploration strategies has been executed 50 times to account for the unpredictability imposed by the optimization procedure.

We compare the three exploration alternatives in terms of i) optimality of results through the distance metric with respect to the optimal solutions (the lower the better) derived by the exhaustive design space exploration and (ii) number of synthesized solutions that indicates exploration's run-time efficiency. Fig. 4.13 shows the distance from the optimal delay-area design point, by varying the number of synthesized designs for each exploration strategy.

As shown, the efficiency of each exploration strategy is layered in different ranging zones. It is clear that the zone of the proposed methodology dominates almost completely both optimization meta-heuristics applied on the full design space variants. For the same or smaller number of synthesized configurations, the proposed exploration delivers design solutions that are closer to the optimal SVM designs, delivering an average distance error 0.001 with a standard deviation of 0.14. The respective average distance values for the steepest descent on the full design space has been calculated to 2.83 (standard deviation: 2.37), while for the genetic optimizer to 4 (standard deviation: 2.47).

It is important to stress out that the two-phase exploration strategy presented in Section 4.4.2 and validated here is successful because the optimizer is able to search for design points within a greatly compact space which includes very effective solutions. *In other words, it is the combined effect of the pruning guidelines and the optimizer that derive an efficient SVM accelerator.* The discrete steepest descent greedy optimizer was utilized as an example meta-heuristic in order to convey that even a simple optimizer is able to locate near optimal design points within the compact design space.

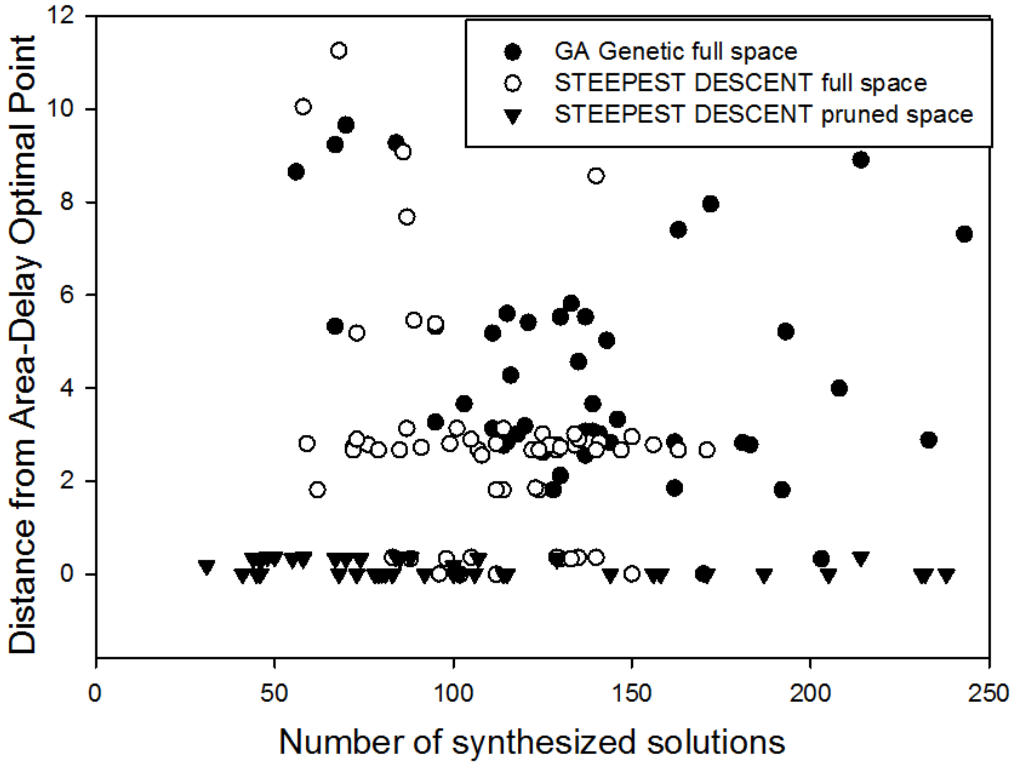


Figure 4.13.: Average Distance from Optimal Design for Different Optimizers

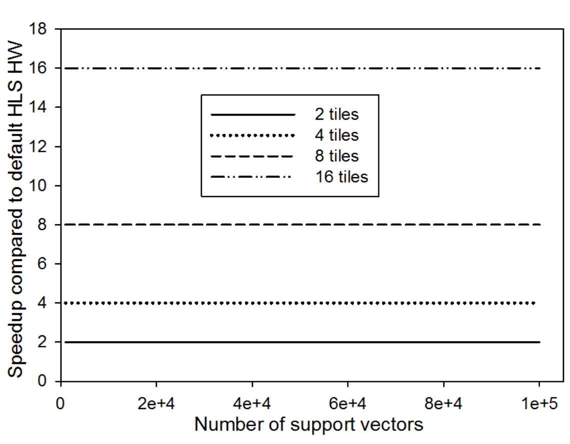
4.6.3. Evaluating derived SVM accelerators classifier at scale

Analysis of our proposed SVM classifier HW acceleration techniques has been presented using a specific SVM model. However, it is important to validate that the proposed work-flow retains its efficiency for SVM classifiers with different characteristics in terms of support vectors number and input feature vector size.

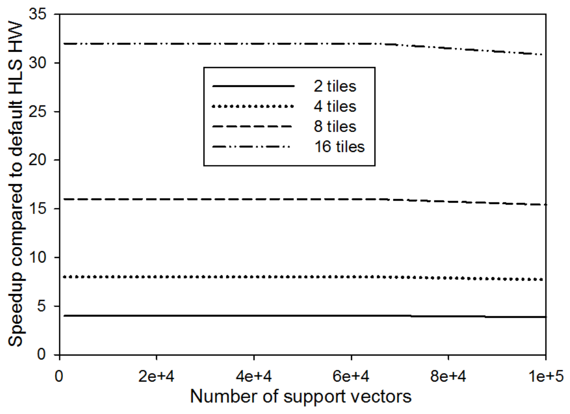
The effectiveness of data level parallelization technique (Section 4.4.1) that partitions the support vector array is tested using support vectors that scale from 1000 to 100000 with fixed feature dimension. Fig. 4.14a shows that speedup remains constant and equal to the number of parallelly executing functional blocks. Scaling is also evaluated in the combination of data-level and instruction-level parallelization technique using an unroll factor of 3. The speedup in that case is doubled and remains constant over different number of support vectors (Fig.4.14b).

Similarly, the effectiveness of the instruction-parallel optimization technique (Section 4.4.1) is tested against support vectors with dimensions scaling from 10 to 1000. It can

[3]The genetic optimizer has been configured with the following parameters: population size: 20, generations: 4

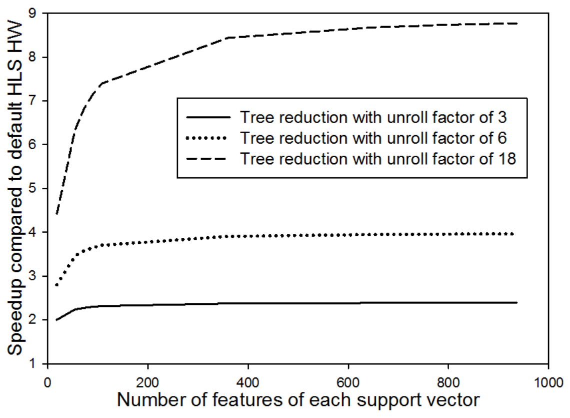


(a) Speedup of tiling technique

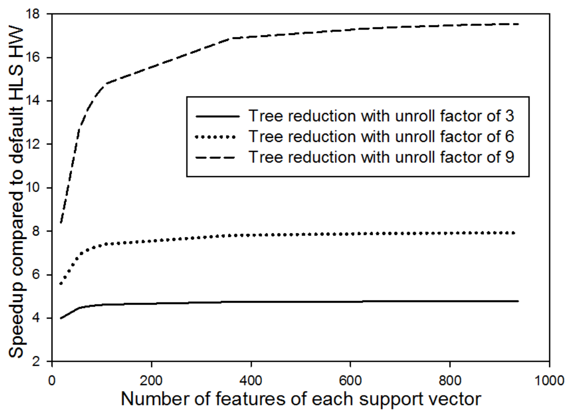


(b) Speedup of tiling combined with tree reduction of unroll factor 3

Figure 4.14.: Speedup of proposed techniques remains the same with scaling N_sv



(a) Speedup of tree reduction technique



(b) Speedup of tree reduction technique combined with tiling of 2

Figure 4.15.: Gain of proposed techniques remains the same with scaling D_sv

be seen in Fig.4.15a that speedup increases until a maximum value is reached and then remains almost constant. The tree reduction technique is also investigated in combination with data level parallelism of a factor of 2. The speedup trend is maintained but its values are almost doubled (Fig.4.15b). This leads to the conclusion that there is no interference in the combination of the two techniques in scaled up versions of the SVM classifier. *In total, results indicate that our proposed methodology retains high latency gains when applied to scaled up SVM classifier models taking into account both increased number of support vector machines number and feature vector size.* To achieve that, resource utilization is significantly increased to support the computational requirement of the bigger model. Inevitably, for high values of the SVM parameters, the utilization exceeds the available resources of the selected target development board. However, this is not a limitation of the proposed methodology but a target HW induced one which is overcome using a larger FPGA in terms of available resources.

4.6.4. SVM based ECG arrhythmia detection

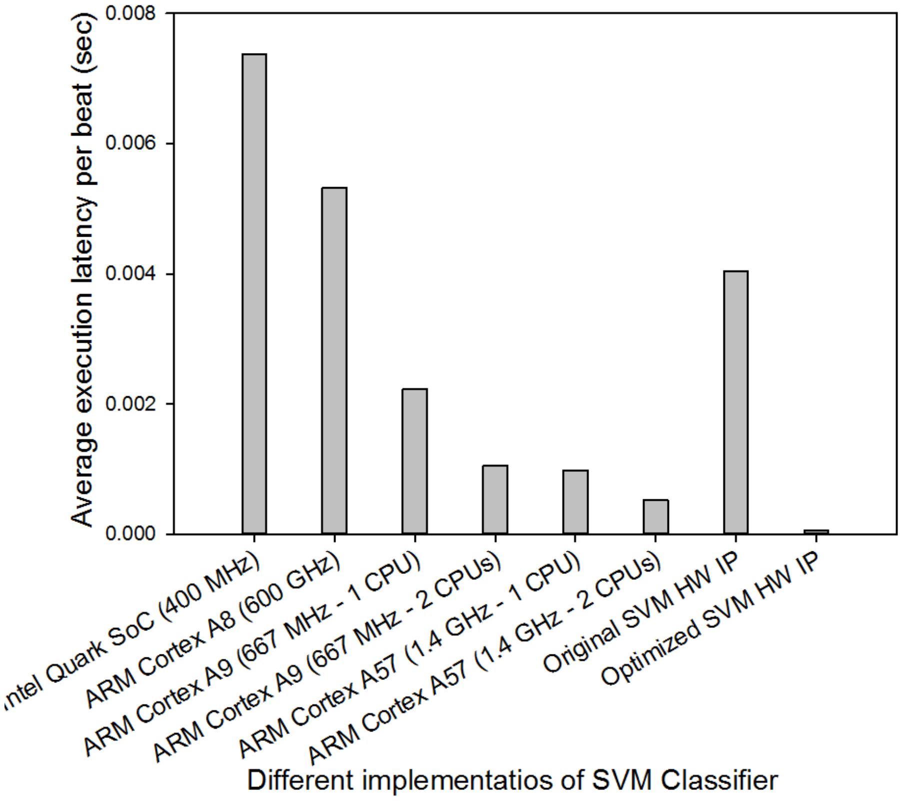


Figure 4.16.: Average execution time per beat

In this section, we evaluate the efficiency of the optimized SVM classifier HW accelerator using a HW/SW co-designed version of ECG based Arrhythmia detection application. A comparison of the diagnosis part of the application is performed, implemented on various target HW platforms with focus on the execution latency of each implementation. The communication latency of providing new input data to the classifier is negligible (ranging from 10x to 900x less than computation time) since its input feature vector consists of only 18 points (Section 4.3.2). For fairness purposes all systems operate on top of a Linux based Operating System and have been compiled using O3 flags of gcc. More specifically the utilized target platforms are:

1. Intel Quark SoC [206] operating at 400 MHz.

2. ARM Cortex A8 [215] operating at 600 MHz.

3. ARM Cortex A9 with 2 processing cores (Zynq Processing System) operating at 667 MHz.

4. ARM Cortex A57 [216] which is a 64-bit CPU with 2 processing cores operating at 1.4 GHz.

5. A Zedboard based HW/SW co-designed system. Its HW IP is derived from Vivado HLS with input SVM source code with no structural modifications or optimization directives applied to it (Maximum Clock Frequency at 100 MHz).

6. A Zedboard based HW/SW co-designed system. Its HW IP is of optimal configuration in terms of execution latency derived from the Pareto optimal points provided by our proposed DSE (Maximum Clock Frequency at 25 MHz).

The testing set is composed of 52291 test vectors, which correspond to feature vectors extracted from heart beats of the MIT-BIH ECG signals. These are provided as input to the different SVM classifier implementations and the average execution latency of the diagnosis stage for all different implementations of the classifier is presented in Fig. 4.16. We observe a correlation between CPU competency and reduced execution latency. In addition, in cases where two processing cores are available, the parallel version of SVM classifier is effective in reducing execution latency. We did not examine the case of more than two workers since the processing cores were fully utilized and thus introducing more working threads could be an overhead. Regarding the co-designed systems, the naive implementation of HLS based SVM HW IP is in most cases not efficient even in comparison to CPUs. On the contrary, *the optimized version of the HW IP, is in every case much more efficient compared to all other design alternatives.* The achieved speedup compared to the unoptimized version reaches up to 78×. Interestingly, the expected speedup values derived from the HLS tool reports is 79.81× which validates that the tool is a trustworthy guide for the designer. The optimized HW IP is also almost 10× faster in comparison to the dual core 64-bit ARM based system.

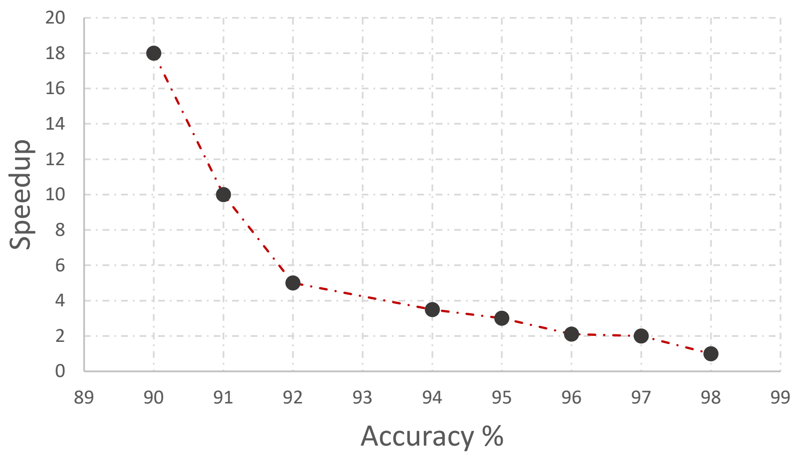


Figure 4.17.: Pareto Front for Accuracy and Speedup Metrics

4.6.5. Performance Evaluation of Approximate SVM

In this section only, the target FPGA and tools differ. Xilinx Vivado-HLS(v2018.2) and the Zynq7 ZC706 Evaluation Board are used to implement all SVM accelerators. The results are based on simulation reports generated by Vivado HLS.

Fig. 4.17 depicts the Pareto Front, which is the output of our methodology. This study explores the trade-off between speedup and classification accuracy when all optimization levels have been applied. Accuracy ranges from 90% to 99% and speedup from 1× to 18×. The highest speedup is achieved for aggressive approximation, i.e. 10% perforation and significant precision scaling. The trade-off of course is reduced accuracy, which is crucial for an application such as arrhythia detection. Fig.4.18 presents the speedup achieved for the fastest configuration resulting by each technique with a target accuracy of 95%. Perforation technique satisfies this constraint for 6% perforation, 1.47× speedup and 96.27% accuracy, whereas precision scaling delivers a configuration with 4× speedup and 97.32% accuracy. Our proposed methodology, delivers a configuration that combines 2% perforation, precision scaling and HLS optimization techniques and outperforms them by delivering 15× speedup and 96.7% accuracy. Finally, Fig. 4.19 presents the resources utilization for the corresponding SVMs. The precision scaling technique is the most efficient one in resources utilization, since reducing the bit width leads to fewer BRAMS for storing the support vectors and smaller-width operators. The small decrease in resources utilization due to perforation, is attributed to storing less support vectors at BRAMS. Our proposed technique shows greater utilization, due to the HLS directives, that require more resources to increase parallelism and dominate over the resources savings achieved by perforation and precision scaling. This increase is compensated by the greater speedup.

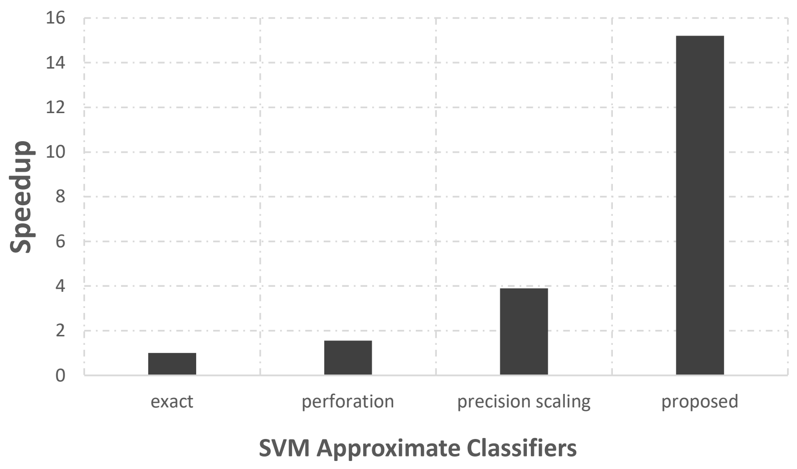


Figure 4.18.: Speedup for Fastest SVM of each technique.

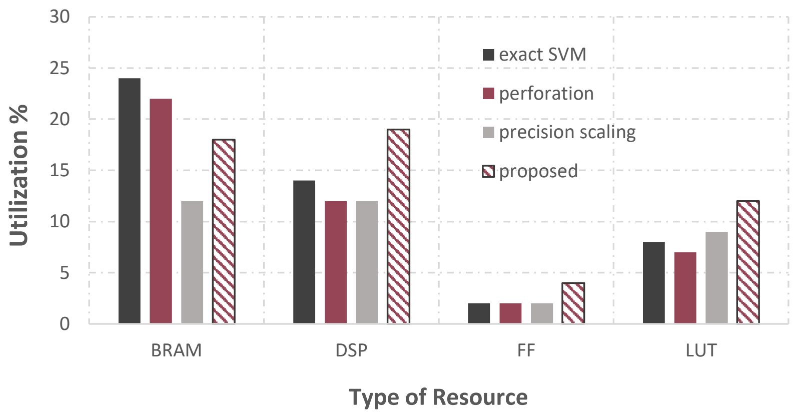


Figure 4.19.: Resources Evaluation for Fastest SVM of each technique.

Chapter 5.

GANDAFL: Dataflow Acceleration for Short Read Alignment on NGS data

Next Generation Sequencing (NGS) technologies have revolutionised genome study through rapid generation of genomics data at low cost. Any genome analysis usually starts with DNA read alignment, employing string-matching algorithms such as Smith-Waterman to compare DNA sequences. The inherent computational intensity and the vast amount of NGS input data the algorithm operates on, create a bottleneck in the workflow. Accelerated reconfigurable computing has been extensively leveraged to alleviate this bottleneck, focusing mostly on high-performance albeit standalone implementations, i.e neglecting the implications of integrating the accelerated functions within the sequencing tools. In existing accelerated solutions effective co-design of the NGS short-read alignment still remains an open issue, mainly due to narrow view on real integration aspects, such as system wide communication and accelerator call overheads. In this chapter, we address the aforementioned inefficiencies and propose ***GANDAFL****, a novel* ***G****enome* ***A****ligNment* ***DA****ta-****FL****ow architecture for SmW Matrix-fill and Traceback stages to perform high throughput short-read alignment on NGS data. We then propose a radical software restructuring to widely-used Bowtie2 aligner that allows read alignment by batches to expose acceleration capabilities. Batch alignment minimizes calling overhead of the accelerators whereas moving both Matrix-fill and Traceback on chip extinguishes the communication data overheads. The standalone solution delivers up to $\times 116$ and $\times 2$ speedup over state-of-the-art software and hardware accelerators respectively and GANDAFL-enhanced Bowtie2 aligner delivers a $\times 1.9$ speedup. This chapter is based on our publications in [217, 218].*

5.1. Introduction

The development of next-generation sequencing (NGS) technologies has dramatically changed the landscape of human genetics research [219]. Advances in the field of DNA and RNA sequencing have led to effective genome mapping and have paved the way to personalized genomic medicine [220]. NGS platforms [26] have now the capacity to generate billions of short fragments of DNA in a matter of hours. These small pieces of DNA, called reads, are the input to various types of genomic analysis such as variant calling [221] and differential gene expression [222]. The first step in any genomic analysis pipeline however is short read alignment, which entails finding a specific location on the reference human genome where a short read is best mapped. The vast amount of sequencing data and the excessive time requirements for this step to execute, have put considerable strain on the computing systems used for genome analysis. Since the throughput of NGS technologies does not cease its exponential growth [223], there is an ever-present need for identifying bottlenecks and proposing accelerated solutions for popular aligner tools.

Several aligners such as BLAST [117], BWA-MEM [112], Bowtie2 [113], Novoalign [224] and CUSHAW2 [225] have been developed that rely on a seed-and-extend model for aligning the short reads. According to this model, in the seeding step, each short read is further fragmented in short pieces, called seeds, that align exactly on the reference genome. In the seed-extension step each seed is extended so that the whole short read aligns with the reference, allowing mismatches. Most of the state-of-the-art aligners implement variations of the Smith-Waterman string matching algorithm [118] to perform the seed-extension step. Smith-Waterman is a dynamic programming algorithm that operates in two stages; the *matrix-fill* stage fills a two-dimensional similarity matrix with score values. Starting at a predefined matrix cell, the *traceback* stage traverses the matrix backwards until it constructs a valid alignment path.

The seed-extension step forms a severe bottleneck in modern sequencing frameworks [226, 227]. Bowtie2 aligner first performs the seeding stage for each read, and then extends the seeds through SmW, i.e. *matrix-fill* and *traceback* Quantitative analysis and detailed profiling of different datasets indicates that the Matrix-Fill stage of Smith-Waterman dominates the execution time of Bowtie2 aligner. Depending on the dataset, SmW can take up to 50% of Bowtie2 execution time (see Section 5.5.1 and Fig.5.18a for more details). However this time is actually distributed among independent Smith-Waterman tasks spanning across all reads. Furthermore, each read alignment can invoke a different number of *Matrix-Fill* tasks, each one followed by a *Traceback* task. An initial naive approach would target *Matrix-Fill* for hardware acceleration to tackle the alignment bottleneck. A straightforward integration of an accelerated *Matrix-Fill* phase of Smith-Waterman though [122], [123], [124] would introduce a huge overhead, due to both the immense amount of the accelerator calls and the transferring of the matrices to the CPU for the traceback stage. In fact, taking into account the time overhead provisioned by each call to the accelerator and the accelerator-CPU transfer time for each matrix, the overall execution time of the aligner can actually be increased. A challenge as such has also been noted by [228] regarding JVM-FPGA communication overhead.

Existing works either propose standalone Matrix-Fill Smith-Waterman acceleration ignoring the traceback stage [122–124] and thus the communication overhead in a real system, or provide an end-to-end hardware implementation of both seed and extend phases [229, 230]. Recently, authors in [231] and [232] have implemented such systems based on the workflow principles incorporated in Bowtie2 and Bowtie respectively. Such systems constitute new tools that introduce a learning curve for experts (e.g. new format, new functionality) and come at the expense of safe-to-use results and advanced visualization analyses provided by well-known and defacto sequencing frameworks such as Bowtie2 [113], BWA-MEM [112]. Accelerating existing NGS alignment frameworks exposes several challenges that can be only highlighted by holistically and carefully profiling and modeling the behavior of both the software and hardware coefficients of the co-desinged sequencer. Up to now, there are only a few software/hardware co-design acceleration works for short-read alignment [233, 234]. Therefore, effective co-design of the NGS short-read alignment still remains an open issue, mainly due to narrow view on real integration provided by existing solutions.

In this paper, (i) we identify the communication overhead that typical co-design approaches introduce and propose an architecture, i.e., GANDAFL that alleviates this problem by moving more computation on chip and restructuring the software code to minimize accelerator calls. (ii) We provide a new dataflow implementation of Matrix-Fill and Traceback, that enables high-throughput processing of an unbound number of streaming short reads to cope efficiently with the exponential growth in NGS data. Task-level parallelism, implemented through an interleaving execution pattern, maximizes the throughput via high utilization of the underlying FPGA. (iii) Through effective parameterization of the proposed architecture, we provide an analytical timing model of the accelerator, which is further evaluated and tuned to minimize prediction error for performance. (iv) Finally, we present a Bowtie2 code restructuring that implements an aggregation-batching strategy and feeds the accelerator in high-throughput streaming fashion with minimized transfer and call overheads. To the best of our knowledge, this is the first work that integrates a hardware accelerator into Bowtie2 rather than building an equivalent aligner from scratch. An additional value of our work is that GANDAFL can be incorporated in any short read mapper that relies on the same seed-and-extend mechanism as Bowtie2 (e.g. BWA-MEM) as long as the necessary source code restructuring takes place.

We evaluate GANDAFL, the proposed accelerator, in a two-fold manner, i.e. as a standalone component and integrated in Bowtie2 restructured aligner. As a standalone component, GANDAFL is compared against a baseline dataflow implementation [235] from Maxeler, an optimized x86 SSE Bowtie2 implementation [113], and state-of-the-art FPGA accelerator GACT in Darwin [36] and achieves ×9.5, ×116 and ×2.13 speedup respectively over the latter solutions. For the evaluation we consider datasets of different error rates and quality. The results show that the integration of the GANDAFL with the restructured Bowtie2 source code preserves the accuracy and manages to deliver up to ×2 speedup depending on the dataset.

The remainder of the chapter is organized as following: Section 5.3 presents the theoretical background of short read alignment in genomic pipelines and focuses on popular Bowtie2 aligner and Smith-Waterman algorithm. Section 5.4 extensively describes the implemented architecture and the flow of data through the pipeline while Section 5.5 presents the insights and modifications of Bowtie2 software architecture to accommodate efficient integration of the accelerator. Section A.4 provides a performance and accuracy evaluation of the dataflow engines and the integrated design and finally Section **??** concludes the paper.

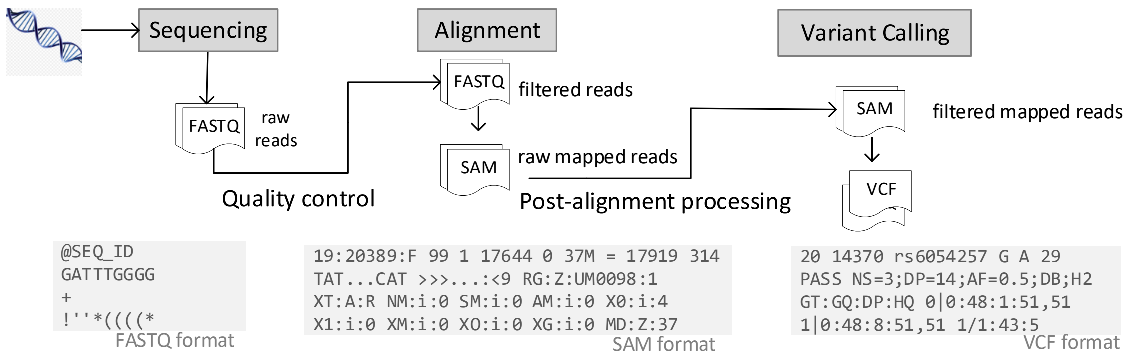
5.2. Related Work

Acceleration of short read alignment has been greatly explored by software solutions such as Edlib [114], WFA [115], and KWS2 [116], which utilize SmW or a similar string comparison algorithm. Custom reconfigurable processors for Smith-Waterman acceleration is an active field of research with several implementation solutions proposed [121]. Most Smith-Waterman accelerators [122], [123], [124], [125], [126] compute the similarity matrix based on a wavefront approach through a pipeline of PEs that forms a systolic array and computes a matrix anti-diagonal per time step. The authors in [124] provide a very detailed architecture as such, that implements a multistage-PE design, and optimize each stage in terms of resources utilization and delay. Similarly in [127], the authors propose a reconfigurable accelerator that implements a modified equation to improve mapping efficiency of a single PE, and a special floor plan to cut down the interface components routing delay. The authors in [128] employ a pipeline of PEs to calculate the similarity matrix but recalculate the matrices for traceback in software for highest-score alignments to avoid memory contention. Although each of these publications suggest optimization techniques on Matrix-Fill, they do not provide a Traceback implementation. In this paper, we extend these designs by implementing the complete Smith-Waterman algorithm along with the Traceback procedure, enriching current literature which is currently lacking in variety of detailed traceback descriptions. In our final high-throughput real system, on-chip traceback diminishes matrix transfer overhead cost and thus enables efficient integration.

There are only a few works of accelerated sequence alignment based on Smith-Waterman with Traceback. The authors in [129] present an alignment engine that performs the traceback in parallel with the matrix fill stage with restrictions on sequence length due to on-chip memory bottleneck. On the contrary, the authors in [130] propose an alignment architecture that accelerates both the forward scan and traceback priotitizing space efficiency for variable reference lengths. The traceback procedure is guided by the host by scheduling multiple partial traceback executions on hardware. Although this accommodates aligning sequences of variable length, it hinders integration with third-party aligners. In contrast to the current work, both accelerators are tailored for long sequence alignment with sequence lengths ranging from 50 to 16000 bases. These sequences are more than an order of magnitude longer compared to those required in short read alignment tasks. Although these works provide the traceback functionality under certain circumstances, they are not designed to cope with a high throughput input rate of short reads generated by an NGS platform. In this work, we attempt to provide an accelerator that continuously accepts new short-reads for alignment and complies with the trends set by latest NGS platforms and state-of-the-art sequencers.

There are also works that target acceleration of widely-used aligners or develop end-to-end hardware implementations of custom aligners from scratch. Authors in [131] emulate the Smith Waterman implemented in BWA-MEM for short-read alignment. The achieved acceleration is extracted from task-level parallelism rather than inner-task parallelism and the focus lies on the scheduling of parallel alignment tasks rather than the specifics of the integration and the handling of traceback intricacies. The work in [132] designs a hardware aligner from scratch based on the algorithm used by BFAST [133]. The implementation requires storing at least 20GB of memory for the genome and a table containing the candidate locations. No information is provided for the implementation of traceback stage.

Recently, Darwin [36] has proposed an end-to-end hardware acceleration for 3rd generation sequencing, implementing accelerators for both seed extract and extend phases and highlighting the importance of including the traceback step on chip so as not to undermine the benefits of hardware acceleration. For the extend phase, the authors introduce GACT algorithm which implements a modified SmithWaterman for arbitrarily long senquencesand is implemented in both ASIC and FPGA. The authors in GenAx [134] propose a highly efficient ASIC accelerator for both seeding and extension step that supports traceback and is based on a finite state automata instead of SmW. The same authors later propose SeedEx [135], an FPGA accelerator for the seed-extension step that targets a cloud FPGA and is also integrated in BWA-MEM aligner. GenASM [136] accelerator is an ASIC accelerator that performs the seed extension step based on Bitap algorithm and accelerates both short and long reads. The authors in [137] also present the ASAP FPGA accelerator for short read alignment. ASAP introduces several modifications to the alignment procedure, that need extra validation before its adoption on existing NGS frameworks, e.g. it utilizes the Levenshtein distance computation rather than SmithWaterman. The latency of ASAP also is dependent on the the number of mismatches between the short read and the reference and supports a simpler constant gap penalty model for the scoring scheme. Our work allows up to 17 edits, higher than the acceptable mismatch rate for most alignments, and handles the affine-gap penalty



```
@SEQ_ID
GATTTGGGG
+
!''*((((*
```

```
19:20389:F 99 1 17644 0 37M = 17919 314
TAT...CAT >>>...:<9 RG:Z:UM0098:1
XT:A:R NM:i:0 SM:i:0 AM:i:0 X0:i:4
X1:i:0 XM:i:0 XO:i:0 XG:i:0 MD:Z:37
```

```
20 14370 rs6054257 G A 29
PASS NS=3;DP=14;AF=0.5;DB;H2
GT:GQ:DP:HQ 0|0:48:1:51,51
1|0:48:8:51,51 1/1:43:5
```

Figure 5.1.: Typical Variant Discovery Workflow.

model.

5.3. Theoretical Background

5.3.1. NGS genomics pipeline

A genome is an organism's complete set of genetic instructions and information. This information is enclosed in regions of the genome, called genes. A genome is essentially a long string of nucleotide bases: adenine, cytosine, guanine and thymine (A, C, G and T respectively). Within a species, the vast majority of nucleotides are identical between individuals. Different expression of genes among individuals create the genetic diversity and lead to a variety of phenotypes. Genome study is essential to discover the mechanisms that lead to this diversity and is of great value to emerging fields like personalized medicine and research for various often incurable genetic diseases. Genome applications that fulfil this purpose are variant calling [162], differential gene expression [163], phylogeny creation [164] etc.

Fig.5.1 illustrates a typical variant-discovery workflow and the most important steps of the workflow. The first step focuses on the generation of the data required for these workflows. The ever evolving genome sequencing technology enables the reading of the sequence of nucleotide bases in a DNA molecule of an individual. This procedure converts raw signals from individuals into short fragments of bases, called short reads. Second generation sequencing platforms [156] generate many millions of nucleotide short reads, with lengths that range between 50 to 300 bp (base pairs). During the generation of short reads, unique sequencing errors and biases are introduced and therefore quality checks are required to identify and correct them. The resulted short reads data are saved in FASTQ format [167].

The next steps. i.e. short read alignment and variant calling, facilitate the reconstruction of the genome of the sample and comparing it to the reference genome of the organism. Short read alignment performs the mapping of the short reads generated in the first step to a location in the reference genome that is most likely its origin. Both reference and read sequences are encoded using numerical values. The short read alignment process results in alignments that can be either perfect matches to the corresponding reference location or include mismatches. Single nucleotide substitutions (SNV), insertion or deletion of a base (indels) are some of the most frequent variants in the sequence of bases that can potentially lead to genetic mutations. Gaps are used to account for insertions or deletions in the sequence, i.e. reference gap and read gap respectively. These variants are called *edits*. After aligning, a Sequence Alignment Map (SAM) file is produced, which includes information on the alignments of reads [168]. A post-alignment processing step is also invoked to correct technical biases, and the corrected alignments are ready for variant calling analysis.

Finally the variant calling step identifies differences between the sequencing reads and the reference genome. It is worth mentioning that, localized realignment is also performed during the variant calling stage to correct artifacts introduced during the alignment phase. The pinpointed variations are reported in an output file in the Variant Call Format (VCF) [169].

5.3.2. Bowtie2 Alignment Algorithm

As shown, short read alignment is at the heart of the genomic pipelines. Several software tools have been developed therefore to perform alignment of shorts reads to the reference genome. Bowtie [170, 171], Bowtie2 [113], Soap2 [172], BWA-MEM [112] are some of the widely used ones. Most aligners adopt a seed-and-extend alignment model to perform the aligment of reads. According to such a model, the seeding phase searches the reference genome for perfect matches of small substrings in the short read, i.e. seeds. This search is sped up through the indexing of the reference genome with various techniques. The exact matches that occur from this search are the candidate positions for the final alignments. The next phase extends the candidates so that the short read aligns against the genome allowing mismatches. The extend step uses dynamic programming to compute score-matrices and construct possible alignments, such as Needleman-Wunsch [177], SmithWaterman [120] and approximate string matching algorithm for Levenstein Distance [236].

Bowtie2 [113] utilizes a pre-built index of the reference genome, which is based on FM-index [88] and BWT transform [174]. The aligner operates on a set of sequencing read files in FASTQ format [168] and outputs a set of alignments in SAM format. Bowtie2 iterates sequentially across the reads of the FASTQ file and aligns each one based on the seed-and-extend model. In the *seed phase*, each read is fragmented into seeds, whose length and interval can be defined by the user [237]. In our implementation seed length is 22 and seed interval is defined by function *$length_{seed}$ = 1 + 1.15 * sqrt($length_{read}$)*. The seeds are prioritized in descending order based on their probability to deliver a valid alignment and create the set of candidates for full-alignment of the initial read allowing edits.

In the *extend phase*, a designated function iterates over the seed hits and initiates Matrix-Fill tasks. Once a seed is selected for further exploration, it is extended into full alignment by performing SIMD-accelerated dynamic programming, i.e. SmithWaterman: Matrix-Fill followed by Traceback. Algorithm 1 presents a simplified version of the complex control that drives the attempts for extending candidate seeds. Multiple conditions between iterations of candidate seeds (line 4) determine if the search for alignments should be continued and if yes, which candidate seeds should be examined. For example, seeds that lay on a location that has already been tested through a previous iteration (i.e. overlapping) will lead to the same alignment and are therefore skipped. Similarly, if several alignments have been found for the read and all the remaining ones are expected to be suboptimal, the search is successfully completed. However, if no successful tries have been made and a limit of failed attempts has been reached, the search terminates. This algorithmic structure spawns an unbound number of alignment tasks per read, each one depending on the previous seed-extension alignment task. Therefore the alignment of a single read is in fact a chain of seed-extend alignments. However, neither the order nor the number of examined seeds can be known a priori.

Each alignment has a score that represents the probability for the read to originate from the corresponding reference's location in the genome. This is an educated guess and Bowtie2 does not guarantee to find the best possible alignment. The found alignment results are sorted based on the alignment score and by default the highest-ranking alignment is reported. The alignment result reported is formed during traceback and includes information about the position of the alignment in the reference genome as well as a list of edits that occur.

5.3.3. Smith-Waterman Algorithm

Smith-Waterman [118] (SmW) is a dynamic programming algorithm for performing local sequence alignment and determining similar regions between two nucleic sequences. The algorithm mainly consists of two phases: (i) filling a similarity matrix and (ii) tracing back the similarity matrix to find the optimal alignment between the two sequences. Let Q, $|Q| = n$ be the read sequence that aligns against reference sequence S, $|S| = m$. Q and S constitute a read-reference pair and alignment task. SmW performs alignment by filling similarity matrix H according to Eq.8.3. In this equation, q and r stand for gap extend and gap open penalties respectively, while sc stands for the substitution matrix that assigns each pair of bases a score for match or mismatch. An alignment score quantifies

Algorithm 1: Bowtie2 Seed&Extend Model

```
1  while not eof in FASTQ do
2      rd = next_read()
3      seed_list = searchAllSeeds(rd)
4      ranked_seeds = rankSeedHits(seed_list)
5      while not done do
6          seed = next_seed(ranked_seeds)
7          (rd,rf) = form_extension(seed)
8          (E,F,H,start) = MatrixFill(rd,rf)
9          if successfull then
10             alignment = Traceback(E,F,H,start)
11             result_list = add(alignment)
12     ranked_res = rank(result_list)
13     report(ranked_res[0])
```

how similar the read sequence is to the reference sequence aligned to. It is calculated by subtracting penalties for each difference (mismatch, gap, etc.) and, in local alignment mode, adding bonuses for each match.

$$\begin{aligned} E_{i,j} &= \max\{E_{i-1,j}, H_{i-1,j} - q\} - r \\ F_{i,j} &= \max\{F_{i,j-1}, H_{i,j-1} - q\} - r \\ H_{i,j} &= \max\{H_{i-1,j-1} + sc[Q[i], S[j]], E_{i,j}, F_{i,j}, 0\} \end{aligned} \tag{5.1}$$

SmW then identifies the highest score in H matrix. Starting at this position, a traceback function traverses the matrices backwards until it reaches a zero-score element and thus acquires the optimal alignment path.

SmW Data dependencies: Fig.5.2 illustrates an alignment example utilizing a simplified linear gap penalty scheme. The match and mismatch bonuses and penalties are defined in the substitution matrix. On the left, Fig.5.2 presents an intermediate snapshot of the calculations whereas on the right the final results. Each cell in the matrix H requires the values from the *up* cells of matrices E, F, H (indicated by blue boxes), the $left$ cells of E, F, H (green boxes) and the $upleft$ cell (red box) of H. Therefore all elements in a single anti-diagonal can be computed in parallel and are dependent only on values from the previous two anti-diagonals. The proposed architecture exploits this property to extract parallelism and thus fills H,F,E matrices per anti-diagonal in n + m -1 steps (details in Section 5.4.1).

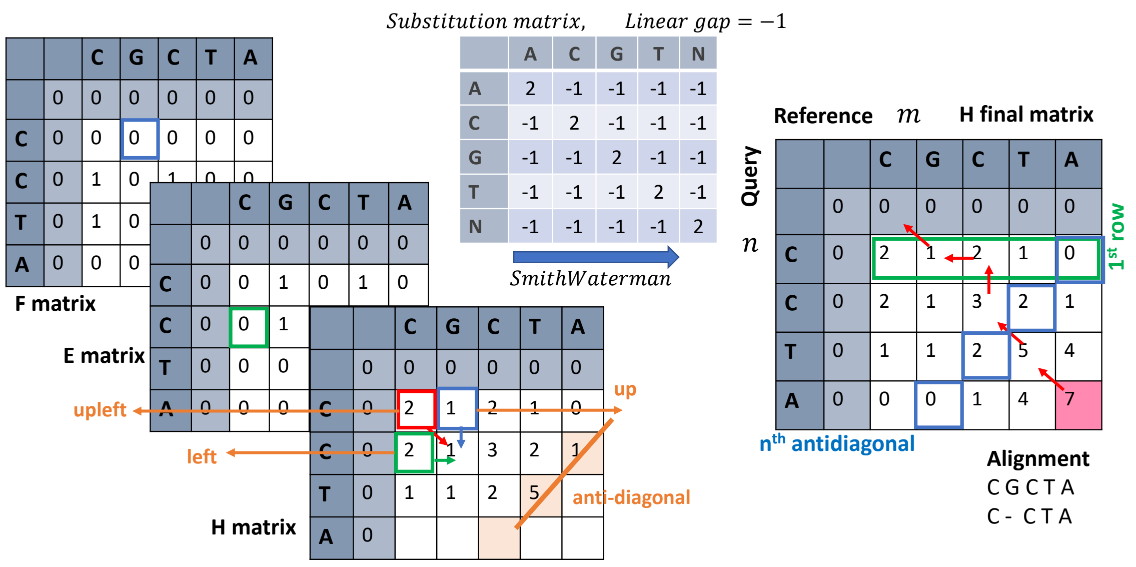


Figure 5.2.: Matrix Fill Dependencies and Traceback example for simplified Smith-Waterman with linear gap penalty scheme.

5.4. Design of the Accelerator System

This work proposes GANDAFL, an architecture of an accelerator system that can be seamlessly integrated into Bowtie2 aligner. Therefore, the input and output interface as well as the alphabet and scoring schema are in compliance with Bowtie2. GANDAFL implements a linear systolic scheme. It includes an additional module that implements the traceback stage of the SmW algorithm (Section 5.4.1), and is enhanced with a series of design optimizations (Section 5.4.2). A detailed description of the optimizations and the synchronization and flow of data through the modules is presented (Section 5.4.3). We also devise an analytical performance model for GANDAFL (Section 5.4.4) and we further discuss its scalability (Section **??**).

5.4.1. Dataflow Smith-Waterman & Traceback Engine

A high-level description of the on-chip architecture is depicted in Fig.A.3. The proposed architecture includes two dataflow modules, each one implementing a phase of SmithWaterman, the *Matrix-Fill* and *Traceback* components. *Matrix-fill* kernel receives as input the read-reference pairs to be aligned. This kernel calculates the matrices H, F, E and the starting point of the traceback procedure through an array of Processing elements. The substitution matrix that defines the match and mismatch penalties is implemented as a Read-Only Memory on chip. The *Traceback* kernel needs to traverse the data generated by Matrix-Fill in reverse order, starting from the aforementioned position. For that

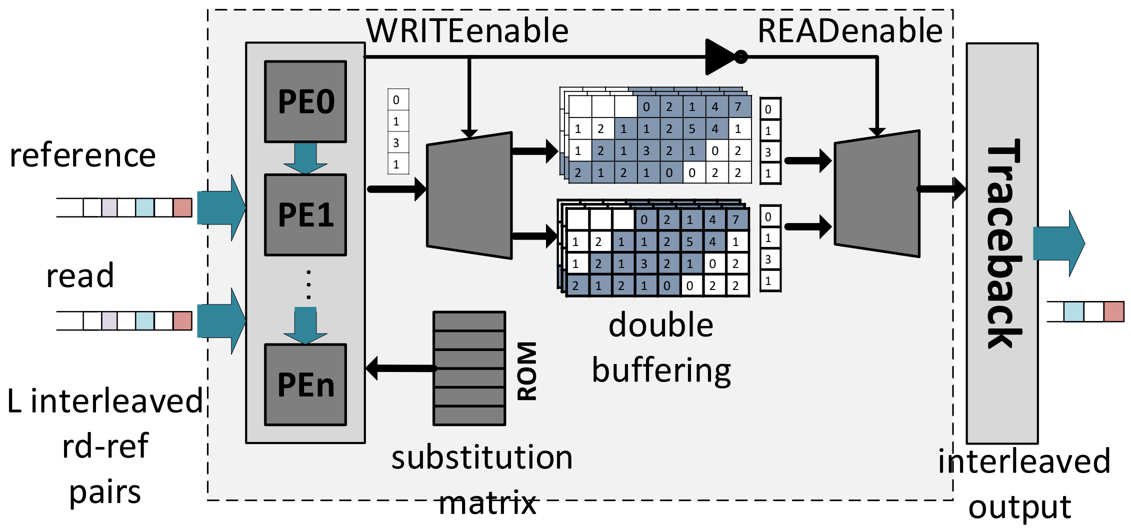


Figure 5.3.: Architecture of Dataflow Engines on Chip.

reason, the matrices are stored on on-chip memory for subsequent use from Traceback. Traceback traverses the stored matrices, finds the alignment for each read-reference pair and fills in buffer with information necessary to the host program to reconstruct the alignment. Detailed description of these kernels and enhancements depicted in Fig.A.3 will be explained elaborately throughout this section.

Matrix Fill

Matrix-fill input. MatrixFill kernel receives as input the read-reference pairs to be aligned. Each pair consists of a read sequence and a reference sequence, which is a candidate fragment of the genome against which the read sequence aligns. Each element in the input sequences represents a single nucleic base of the sequence. The kernel receives a number of read-reference pairs, i.e. a read-reference pair batch. The different read sequences are arranged consecutively in the *read* input stream. The corresponding reference sequences are placed in the same order in the *reference* input stream.

Processing Elements array: The architecture is built on an array of Processing Elements *(PEs)*, equal to the length n of Q read sequence. Each PE_i holds a single base character of the read sequence and computes the i^{th} row of E, F, H matrices, as reference sequence S of length m is streamed through it. Fig.5.4 illustrates matrix H computation for the pair of sequences $\{Q_1, S_1\}$ of lengths $n = 4$ and $m = 5$ respectively. Each time step L in the time axis of the figure corresponds to the time required for the computation of a single cell value and therefore an antidiagonal. The PEs operate in parallel and each one computes

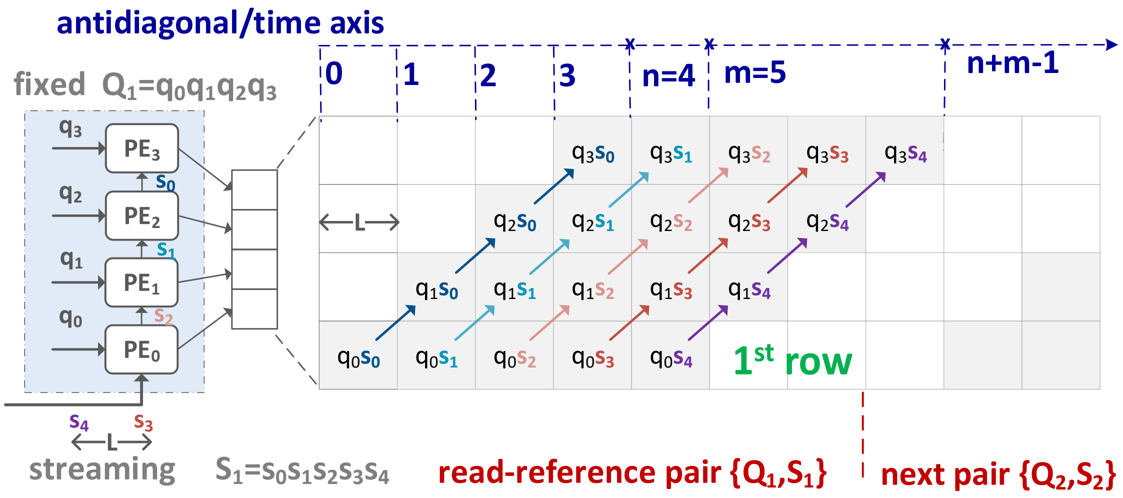

antidiagonal/time axis
fixed $Q_1=q_0q_1q_2q_3$
0
1
2
3
n=4
m=5
n+m-1
PE_3
PE_2
PE_1
PE_0
q_3
q_2
q_1
q_0
s_0
s_1
s_2
s_4
s_3
L
streaming
$S_1=s_0s_1s_2s_3s_4$
q_3s_0 q_3s_1 q_3s_2 q_3s_3 q_3s_4
q_2s_0 q_2s_1 q_2s_2 q_2s_3 q_2s_4
q_1s_0 q_1s_1 q_1s_2 q_1s_3 q_1s_4
q_0s_0 q_0s_1 q_0s_2 q_0s_3 q_0s_4
1st row
read-reference pair $\{Q_1,S_1\}$
next pair $\{Q_2,S_2\}$

Figure 5.4.: Example of computation of H matrix by PE array unfolded in time for sequence lengths $n = 4, m = 5$.

an element of the current antidiagonal. Each PE_i starts receiving reference bases at time i. A new s_i base arrives every L cycles. For example, by time $t = n = 4$ base s_0 has passed through all PEs, bases s_1, s_2, s_3 have reached PE_2, PE_1, PE_0 respectively and s_4 has not yet entered the pipeline. In time step $t = m = 5$, all reference bases have passed through PE_0 and therefore row_0 of matrix H has been computed. Similarly, the rest of the rows are computed by other PEs. When all S bases stream through all PEs (after $n + m - 1$ steps), all values of similarity matrices are computed.

Fig. 5.5 depicts the schematic diagram of the architecture inside a single PE. The schematic implements equation 8.3 as a chain of comparators, which propagate the maximum value of their inputs until $H_{i,j}$ is computed. Each PE_i is assigned with a base character q_i of the query sequence. This value is retained until all reference characters s_k are streamed from PE_{i-1} through PE_i and into PE_{i+1}. Characters q_i, s_k are used to index a ROM (scoring matrix) and fetch their match or edit score. The computation of an $H_{i,j}$ cell takes L cycles according to circuit timing. Values of $E_{i,j-1}, H_{i,j-1}$ were last computed by the same PE_i and therefore they are stored for L cycles before being fed back into the same PE for $H_{i,j}$ generation. Values $F_{i-1,j}, H_{i-1,j}$ were computed by the previous PE, PE_{i-1}, and are stored for use by PE_i for L cycles after their generation. Similarly, $H_{i-1,j-1}$ is computed by PE_{i-1}, L cycles before $H_{i-1,j}$, so it is stored for $2 * L$ cycles before being consumed by PE_i. Fig. 5.5 includes buffers of length L to illustrate the delay and storing of values until each is ready for consumption. The last PE runs additional logic to locate the cell with the maximum score in the final row.

MatrixFill output. Due to anti-diagonal dependencies, matrix elements are stored per antidiagonal as soon as they are computed, and not per row as expected. This leads to the

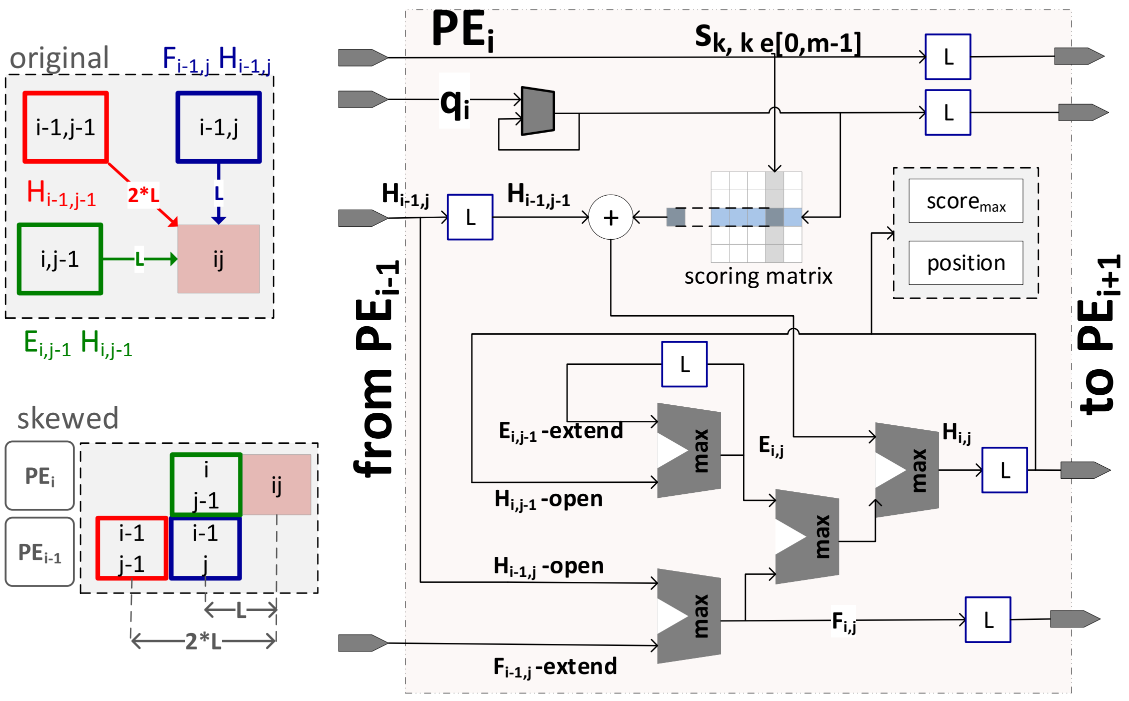


Figure 5.5.: Schematic diagram of PE architecture and flow of Data between consecutive PEs.

skewed matrix pattern depicted in Fig.5.4. Each row of the skewed matrix is implemented as a vector of n elements, so that all PEs can write simultaneously the cell values generated per time step. The row dimension is naturally equal to the number of antidiagonals, i.e. the time steps required for filling the matrix. Therefore the buffer is implemented as a two-dimensional matrix with $n + m - 1$ rows and n row-length.

Traceback

Traceback module reconstructs the alignment path by traversing the H matrix in reverse order, as depicted earlier in Fig.5.2(b). Matrix fill streams the position of the maximum score to the Traceback module and buffers out E, F, H column-vectors, i.e. antidiagonals, in reverse order from the $n+m-1^{\text{th}}$ to the 1^{st} one. The maximum score can be any of the elements of the n^{th} row of H matrix, and therefore it is not necessarily found on the first antidiagonal received by Traceback. For that reason, the alignment path computation begins when the anti-diagonal containing the starting point arrives. Based on the values of E, F, H received, bases $\{q, s\}$ and the scoring scheme, the backwards step is resolved and it could either be the *up*, $left$ neighbor of any of H,F,E matrices, or the $upleft$ element of matrix H.

Fig.5.6 demonstrates how Traceback receives one column, and thus an anti-diagonal, of

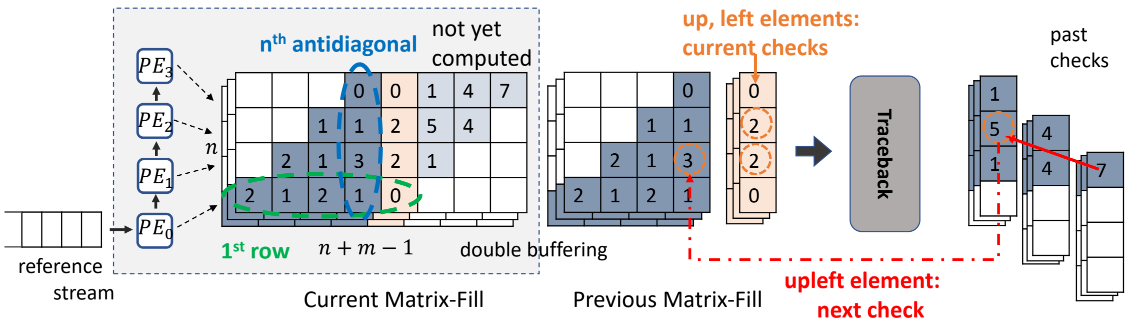


Figure 5.6.: Flow of data from Matrix Fill to Traceback phase.

each matrix per L cycles for $n = 4, m = 5$. At this time, Traceback has received 3 antidiagonals and resolved the first backwards step. Based on the current cell of the 3rd antidiagonal, Traceback deduces that it has to move to the upleft cell, which is 2 antidiagonals away and therefore has to wait for $2 * L$ cycles to receive it.

As the traceback progresses, potential edits, between the two sequences occur. Each edit is fully described by its *type*, the *position* it occurs on the reference sequence, and the bases of the sequences at this position $qbase, sbase$. The types of edits can be one of the: substition, read gap, reference gap and are encoded as an integer number. Our design adopts an upper limit of edits, i.e. 17 $MAXEDITS$, which is in compliance with the range of edits for short reads generated by modern sequencers [238]. This is verified in Fig.5.7, which shows that 99.99% of Bowtie2 found alignments (reported and not) for our utilized datasets (see Section 5.6.1) have less than 18 edits.

Traceback output. The output of Traceback is constructed in compliance with the data required by Bowtie2 to allow for seamless integration and reporting of alignments. Such data are a *flag* indicating success or failure, the alignment *score*, the *cell* with the max score, the number of edits and the start of the alignment on the two sequences ($pos1, pos2$). For each alignment pair, these values are streamed to the host along with a list of the edits ($type$, $position$, $qbase, sbase$). Table 5.1 summarizes the input, output and intermediate data of the dataflow engines.

5.4.2. Streaming Optimizations

In order to minimize stalls in the dataflow pipeline due to intra- and inter-data dependencies, the following streaming optimizations have been adopted:

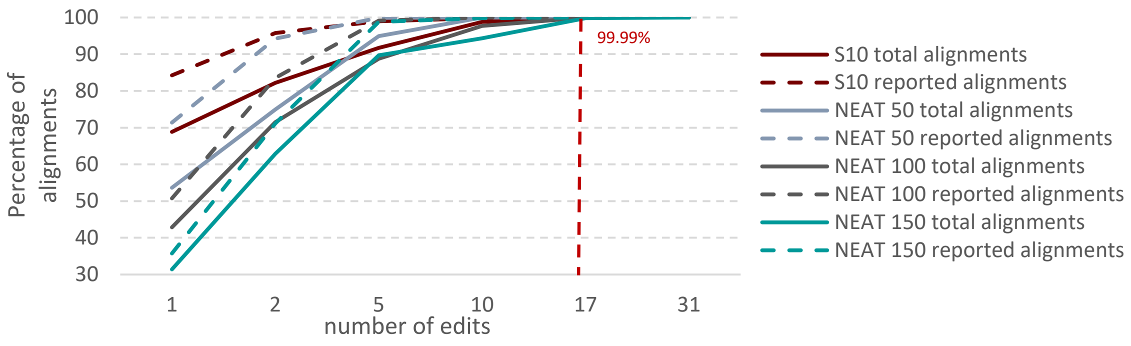


Figure 5.7.: Cumulative distribution of edits for all utilized datasets.

Interleaving Data Scheme

The computation of a cell value from each PE introduces a latency of L clock cycles between the computation of consecutive anti-diagonals. To avoid idle clock cycles due to this intra-data dependencies, the computation of L read-reference pairs is interleaved.

Input sequences interleaving: According to this technique elements of subsequent read-reference pairs are interleaved into a single sequence. Fig.5.8 illustrates the interleaving scheme for $L = 2$ read-reference pairs. Reference sequences S_1, S_2 are combined in a single input buffer that contains double the elements than each sequence on its own. The bases of the original sequences are placed in the buffer in a round robin manner, i.e. alternately in the case of $L = 2$. Between consecutive elements of a sequence S_i there is an offset of $L-1$ elements of other reference sequences. For $L = 2$ this offset equals 1. The interleaving of the read sequences is implemented in the same way.

Output buffers interleaving: The interleaving processing also affects the layout of all intermediate and final output buffers, i.e. E, F, H matrices and Traceback results. E, F, H

Table 5.1.: Input, Output and Intermediate data in the Dataflow Engines.

MatrixFill		Traceback
input	intermediate	output
$S(char)$	$E(vector(n))$	$fail, score, subs, gaps(char)$
$Q(char)$	$F(vector(n))$	$rpos, qpos(char)$
	$H(vector(n))$	$rposini(int16)$
	$positions(uint16)$	$edit_{pos}, edit_{type}(char)$
	$S(char)$	$edit_{qchr}, edit_{chr}(char)$
	$Q(char)$	

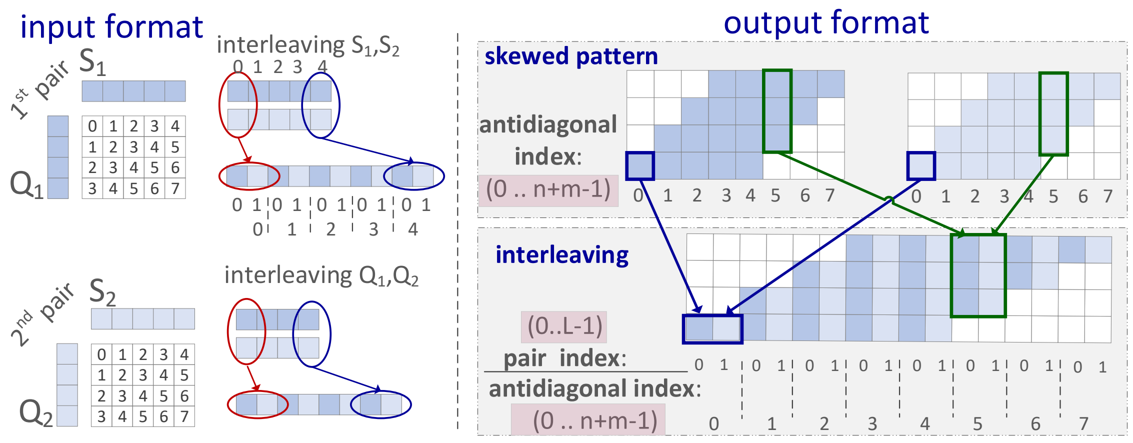


Figure 5.8.: Example of interleaving technique for L=2, reflected in both input and output buffers.

matrices follow a skewed pattern due to antidiagonal dependencies as explained in Section 5.4.1. Fig.5.8 demonstrates how the final matrix layout is generated when applying interleaving on skewed matrices of independent alignment pairs $\{S_1, Q_1\}$ and $\{S_2, Q_2\}$. MatrixFill first computes and writes data for the first antidiagonal of pair $\{S_1, Q_1\}$ and then moves on to the "first" antidiagonal of pair $\{S_2, Q_2\}$. The rest antidiagonals follow in the same manner. Therefore, the final matrix is arranged in groups of antidiagonals of the same order, each from a different task. The groups can be accessed through the $antidiagonal - index$ in Fig.5.8. Within a group, the offset of the antidiagonal of a specific pair can be accessed through the $pair - index$. This can be generalized for L interleaved pairs. Interleaving technique allows the computation of L anti-diagonals per L cycles, instead of 1 per L clock cycles and therefore greatly improves the throughput of the design.

Taking into account the interleaving optimization technique, the final buffer size is finalized and described in the following equation.

$$matrix_{dim} = ((n + m - 1) * L) * n$$

The interleaving technique is also reflected in a straightforward manner in the output buffers of $Traceback$, where the output of each independent pair can be extracted utilizing the double index addressing scheme.

Double Buffering Technique

Traceback starts operating on the last values generated by Matrix-Fill. Therefore, Traceback module has to wait for Matrix-Fill module to complete writing the matrices. Similarly, Matrix-Fill has to wait for Traceback to read all required values from the matrices before aligning the next batch of L read-reference pairs and writing over the data. To avoid halting Matrix-Fill's operation while streaming data to Traceback, the *double buffering* technique is employed.

Implementing the double buffering: This technique is applied to all matrices E, F, H. Two copies are allocated for each matrix and Matrix-Fill and Traceback write and read data, respectively, alternating between the two on-chip instances of each matrix. Fig.5.9 demonstrates the parallel read and write operations performed on the two instances of each matrix during the alignment of a single L-batch. A control signal is utilized as the selector in a multiplexer and defines which of the two allocated memories is being written by Matrix-Fill. The complement of this signal is the selector of a multiplexer that decides from which memory to fetch the data read by Traceback. The write and read enable signals swap values for the aligment of the next L-batch. The two states depicted in the figure are alternated until all bathes are aligned.

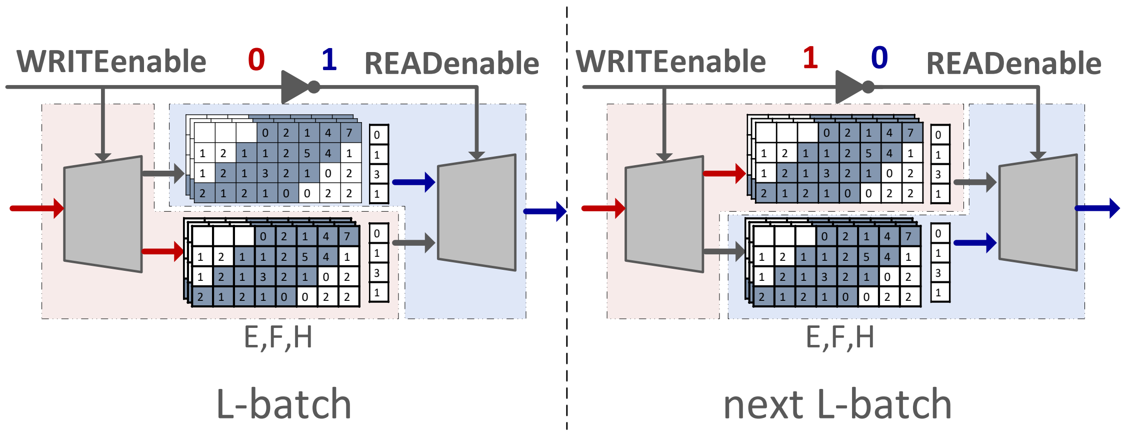


Figure 5.9.: Alternate READ/WRITE operations based on Double Buffering technique for consecutive batch alignment tasks.

5.4.3. Control and Data Flow of Engines

Matrix-Fill and Traceback engines are carefully synced and orchestrated to operate in a pipelined manner. The end-to-end pipeline is build on set of *modules*, synchronized through a set of event-driven counters that are a mathematical expression of the clock cycles. These counters schedule the flow of data so that they pass through modules in

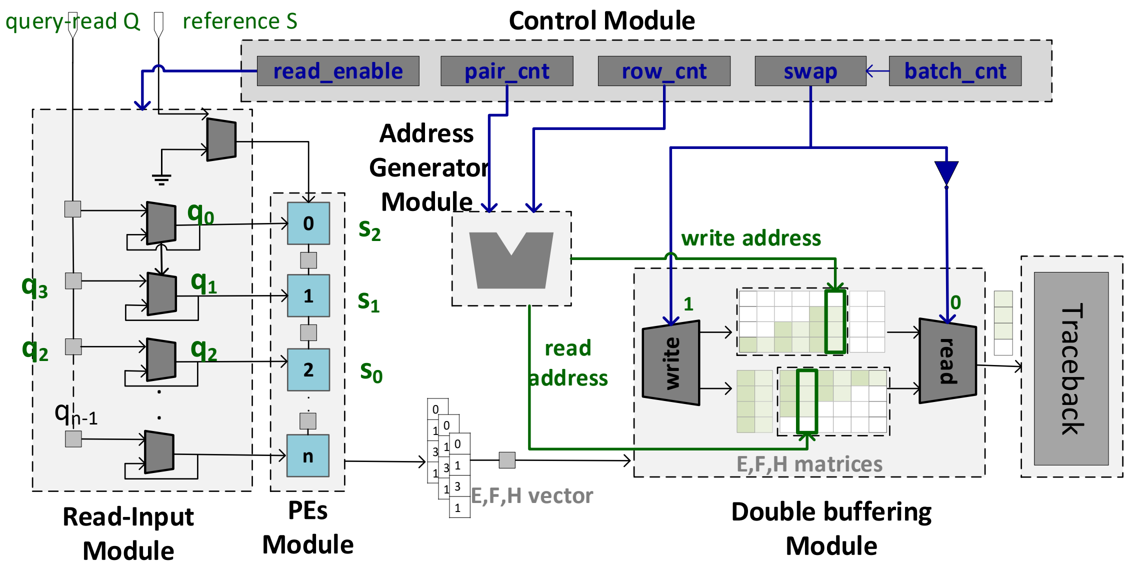


Figure 5.10.: Control (marked blue) and Data Flow (marked green) within the pipeline. The state of each module for aligning 2 batches with $L = 2, n = 4, m = 5$ at time $\{batch\_cnt, row\_cnt, pair\_cnt\} = \{1, 2, 1\}$ is illustrated.

designated time intervals in order to generate the final results. Fig.5.10 illustrates the pipeline of modules and the flow of data when aligning two batches of read-reference pairs with $L = 2, n = 4, m = 5$.

There are three basic counters, the *pair_cnt*, the *row_cnt* and the *batch_cnt*, respectively. The *pair_cnt* is a modulo counter that counts the cycles required for the computation of a single antidiagonal. Its maximum value is equal to L and it essentially counts the pairs interleaved. In Fig.5.10 this counter has reached its maximum value 1, indicating that an antidiagonal of the second pair is being computed. When this counter wraps, it triggers an increase in the value of the *row_cnt*, which corresponds to the index of the anti-diagonal being computed until all $n + m - 1$ of them are ready. For example, current *row_cnt* value in the figure is equal to 2, therefore the third antidiagonal is under computation. ach time the *row_cnt* overflows, the *batch_cnt* increases and signals the alignment of a new batch of L reads. Fig.5.10 has *batch_cnt* equal to 1, hence it depicts the alignment of the second batch.

These counters formulate control signals $read\_q, read\_s$ for the *Read Input module*, which is responsible for feeding read and reference sequences into the *PEs module* within the time frame of an alignment task of a single L-read batch. Every L cycles, in case of Fig.5.10 every 2 cycles, a new reference character of a single pair is propagated through the array of PEs. In the meantime, reference characters of other pairs stream through. Fig.5.10 showcases that by this time s_0 of the second pair has reached PE_2 whereas s_3, s_4 have not entered the PEs array. Characters of the first pair are held in intermediate buffers for use in the next cycle. Similarly, the read sequences pass through the array of PEs. However, each character is assigned to a single PE and its value is retained until the read's paired reference sequence passes and a row of the matrix is computed. For example, in Fig.5.10 PE_2 updates its q_i value with q_2 while PE_1 holds on to q_1 and ignores streaming value q_3.

The operation of *PEs module* is also driven by the counters. Each PE computes L elements every L cycles and completes L alignment tasks when *row_cnt* overflows. The *PEs module* depending on the value of the respective counters computes the elements of the $row\_cnt + 1^{th}$ E,F,H anti-diagonal of the $pair\_cnt + 1^{th}$ read of the $batch\_cnt + 1^{th}$ L-read batch. For example, in Fig.5.10 the 3rd antidiagonal of the 2nd pair of the 2nd batch is being computed.

PEs Module writes on one copy of the matrices and streams out the second one to Traceback. The *Double Buffering Module* defines which matrix is being written by the PEs during each batch alignment task through the *swap* signal. The "swap" signal alternates between values 0 and 1, triggered by a new batch of L interleaved pairs. In other words, when the *row_cnt* overflows, it signals the beginning of a new alignment and triggers the complement of the swap value.

The matrices read/write operation alternate for each batch but they share the same layout and addressing schema. The *address generator* module constructs the addresses for matrices E, F, H based on the values of the counters. Fig.5.11 illustrates the addressing schema described in the following equations, applied in combination with the double buffering technique for each matrix.

$$write\_address = row\_cnt * L + pair\_cnt$$

$$read\_address = ((n + m - 1) * L - 1) - row\_cnt * L + pair\_cnt$$

The write address indicates which row of each matrix is currently being written. Similarly, a read address is generated, that indicates which row of a matrix is streamed to Traceback. Fig.5.11 on the left shows the values of write address while the matrix is filled in from top to bottom for $L = 2, n = 4, m = 5$. At the same time, the alternate matrix is read in reverse order from bottom to top. Fig. 5.10 also illustrates which row is written at the given time instance and which row of the alternate matrix is streamed out.

The *Traceback module* constructs the alignments of an $L - batch$ and identifies the edits. Traceback receives row by row the matrix written during the previous batch alignment and decides which aligment path is the optimal one. In Fig.5.10 addresses $\{15\ \textit{down to}\ 13, \textit{and}\ 11\}$ have already been received by Traceback and address 12 is cur-

Table 5.2.: Formulas for Modeling Time Analysis.

	Cycles	Latency
Matrix Fill (MF)	$(n+m-1)*(batches+1)*L$	$cycles_{MF}/f$
Traceback (Tr)	$(n+m-1)*(batches+1)*L$ $+MAXEDITS*L$	$cycles_{Tr}/f$
T_{comp}	$\max(t_{MF}, t_{Tr}) = t_{Tr}$	
	Bytes	Latency
Input data	$b_S + b_Q =$ $(n+m)*batches*L$	$bytes_{in}/BW$
Output data	$b_{edit_{feats}} + b_{alignment_{feats}} =$ $4*MAXEDITS*batches*L$ $+10*batches*L$	$bytes_{out}/BW$
T_{comm}	$\max(t_{input}, t_{output}) = t_{input}$	
$T_{exec} = \max(T_{comp}, T_{comm}) = T_{comp}$		

rently being streamed. Within Traceback, three rows $\{15, 13, 11\}$ of pair 1 have been processed, whereas only pair 2 is waiting for its third one.

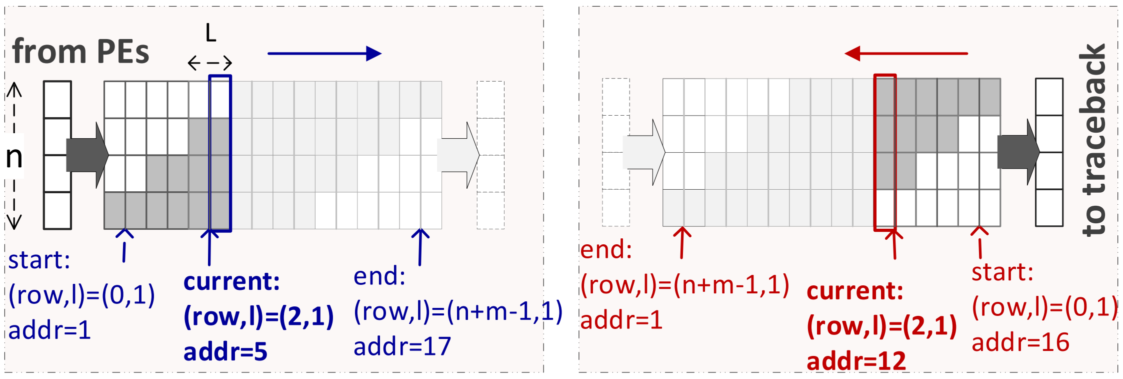


Figure 5.11.: Address generation and matrix indexing applied on double buffers .

5.4.4. Analytical Performance Model of the Accelerator

The parameters describing the system can be used to formulate a timing model. The dataflow computing model allows the programmer to create signals that control the flow of data through the instantiated hardware. As described in Section 5.4.3, the control signals count clock cycles and their overflow value represents the depth of the pipeline. Therefore the programmer can compute the latency of the pipeline as a function of the dimensions of the input dataset, the size of the input and the clock frequency. This function computes the exact number of cycles required for the engines to run.

In the streaming dataflow model implemented in this case, the computation is overlapped with the transfer of data between the host and FPGA. In the dataflow engine, the computation starts as data start streaming in the accelerator and when the first results are ready, they are streamed out. Hence, the stages "sending input data", "computation" and "sending output data" are executed in a pipelined manner. Therefore, the total execution time is the maximum of the computation time (T_{comp}) and the data transfer time, i.e. the communication time between host and FPGA (T_{comm}). The communication time can be calculated as the maximum data size transferred in either direction divided by the bandwidth. The bandwidth depends on the physical interconnect, i.e. in this case PCIe. The computation time is equal to the maximum of the execution latency of dataflow kernels $MatrixFill$ and $Traceback$. Table 5.2 includes the formulas that estimate the time steps required for each operation.

$$
\begin{aligned}
T_{comm} &= \max(t_{inputdata}, t_{outputdata}) \\
T_{comp} &= \max(t_{MatFill}, t_{Tr}) \\
T_{exec} &= \max(T_{comp}, T_{comm})
\end{aligned}
$$

MatrixFill & Traceback: A single $MatrixFill$ task runs for $(n + m - 1) * L$ cycles. $MatrixFill$ executes as many times as the number of batches for alignment and $(n + m - 1) * L$ extra cycles for sending data to Traceback, i.e. $(n + m - 1) * L \times (batches + 1)$ cycles. $Traceback$ also runs for extra $(n + m - 1) * L$ cycles in the beginning to store S, Q sequences and for extra $MAXEDITS * L$ cycles in the end to send back data to the host. Therefore it dominates over $MatrixFill$ and determines the T_{comp} time.

Input & Output data:

The data send into the FPGA are the read and reference sequences of lengths n and m respectively. The output data include the information on the location and the type of the found alignment ($10*batches*L$ bytes) and the potential edits ($4*MAXEDITS*batches*L$ bytes) of each alignment. The transfer time is calculated as the fraction of the bytes and the bandwidth. The input data bytes are more than the output bytes based on the equations and the parameters value, therefore the time to stream input data dominates. Similarly, the computation time dominates over the communication time and therefore the final estimated execution time is defined by T_{comp}.

The estimated execution time is compared to actual measurements on the FPGA with dataset input size scaling from 30 to 100 million input reads. As illustrated in Fig.5.12, the initial model follows the trend of the real measurements, however the deviation is great and the root mean squared error is 13.5 seconds. This deviation is due to neglecting the delay for data transfer to and from the FPGA. To tackle this, we perform a linear regression on the measured error values and generate a linear model for the er-

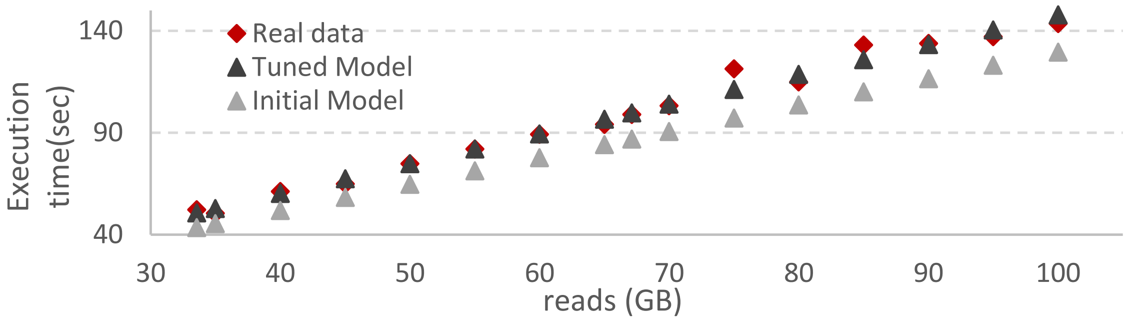


Figure 5.12.: Accuracy of Performance Model based on static timing analysis.

ror prediction. Adding this correction term to the initial model, delivers a tuned model with minimized deviation, root mean squared error of 3.7 and mean absolute error of 2.5 seconds.

Efficiency in terms of PE utilization: As shown in Table 5.2, execution time of the accelerator, i.e. T_{comp}, has a linear relationship to the number of *batches* $\times$ the length of a single batch L. Therefore, in order to examine the utilization percentage of a PE, we focus on a single batch run. Based on the timing model presented in Section 5.4.4, each PE is active as long as a reference sequence passes through it. Assuming as n the length of the read sequence and as m the length of the reference sequence, the percentage of time a single PE is active is described by the following formula:

$$\frac{m}{n+m-1} * 100\%$$

This formula can be easily extracted from Fig.5.13. For *read length* $n = 100$ and $reference\ length\ m = 160$, this percentage translates to 61.8%. With minor changes to our architecture and scheduling scheme (e.g. adding extra memory for matrices E,F,H and changing the start and ending values of the counters presented in Section 5.4.3) we could also pipeline the execution of batches. This way we could achieve almost 100% utilization for each PE. However, this comes at the expense of more BRAM utilization, as two MatrixFill modules would overlap for several cycles and require write access to the same E, F, H instance. A third copy for part of matrices E,F,H would therefore be required to resolve "read after write" dependencies As BRAM utilization is the critical resource in our design, we did not pursue this implementation.

5.4.5. Scalability of Design

GANDAFL can scale in two directions to exploit the full extent of platform capabilities. First, we explore the increase of the depth of the pipeline to address short read alignment problems of different sizes. Secondly, we explore horizontal scalability, i.e. we replicate

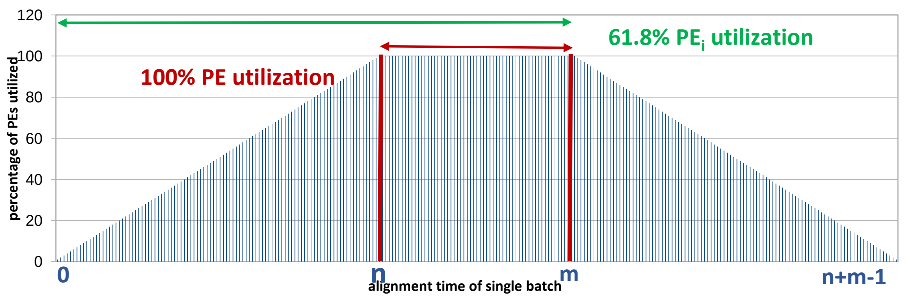


Figure 5.13.: PE utilization in time.

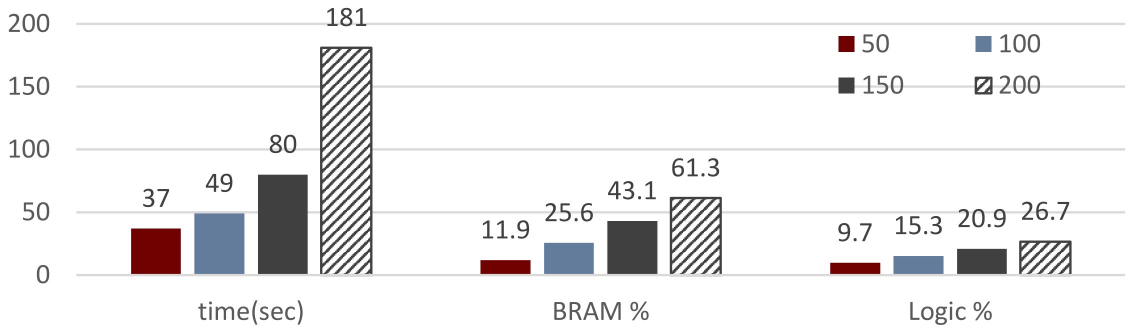


Figure 5.14.: Performance and Area utilization for Various Lengths of Read.

the single instance of the design and map it multiple times until we reach the limit of available resources within a single device or use up all available connected FPGA devices.

Variable Sequence Lengths

Depending on the sequencing technology and application, the length of the short reads can vary. For example, for Illumina sequencing, length ranges from 50bp to 300bp. Based on the equations presented in Section 5.4.4, the design of the engine is fully parameterizable and therefore supports various alignment problems. The length of read and reference sequences are both parameters in the accelerators design, both required in compile time. Read length defines the length of the PE array. Both read and reference length define the depth of the implemented BRAMs.

Fig.5.14 includes utilization results and execution latency for varying read lengths. Fig.5.14 indicates that BRAM resources increase almost linearly with the increase in read length as opposed to execution time. When scaling input length from 50 to 100 bp the increase in latency for 30 million reads is $\times 1.34$ whereas from 50 to 150 bp is $\times 2.17$. Latency escalates for 200bp length, due a drop in frequency from 200 to 100MHz.

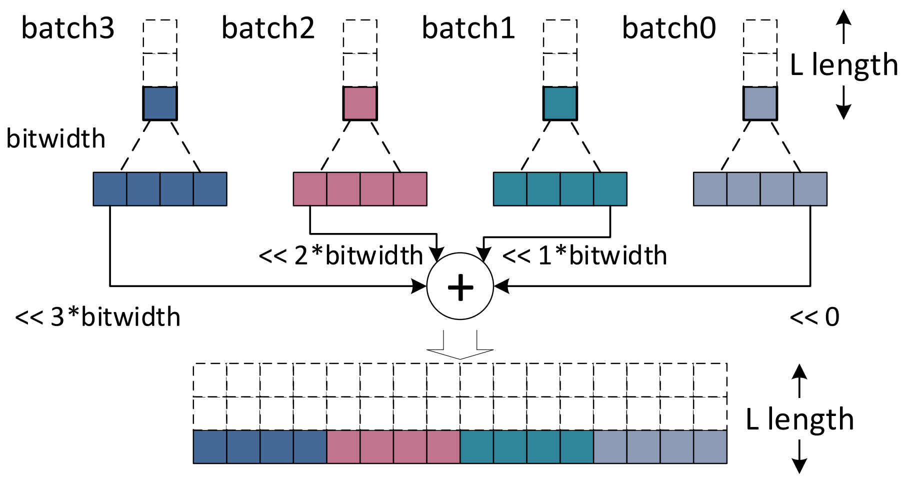


Figure 5.15.: Multiplexing multiple input batches into one batch for multi-instance design.

Multiple Instances - Single Device

Given an alignment task with fixed lengths for read-reference pairs, multiple MatrixFill & Traceback engines can be instantiated in order to fully exploit available resources within a single FPGA board and enable parallel processing of read-reference pair batches. Multiple read-reference batches are combined into a single batch by merging the input and output streams. For that purpose, the elements of each input stream have a wider bit-width, equal to an integral multiple of the *bitwidth* of the respective stream of the single-instance design. This integral multiple equals the number of engine instances. Therefore, each element now contains multiple nucleic bases, as many as the instances, each belonging to a different sequence pair. The initialization of these streams takes place on the CPU. Fig.5.15 illustrates how n-bit-wide input read sequences from 4 batches are multiplexed into a single $4 * n$-bit-wide stream.

Each input stream is subsequently decoupled in as many streams as the number of instances. The resulting streams are of equal length and their elements have the bit-length of the respective stream of the single-instance design. The decoupling is performed by an additional *demux* kernel instantiated on hardware. The *demux* kernel receives as input the multiplexed streams, decouples each element utilizing bitwise-operations and propagates them to the corresponding instance. The instances operate in parallel and the results of each one are multiplexed into streams of wider bit-width by a respective *concat* kernel. This kernel shifts the individual result of each instance into the respective bits and concatenates all of them into a single result.

Table 5.3 presents resources utilization and achieved clock frequency when implementing only MatrixFill and MatrixFill & Traceback engines. The number of instances implemented in both cases is restricted by BRAM utilization. Fig. 5.17 presents scaling of

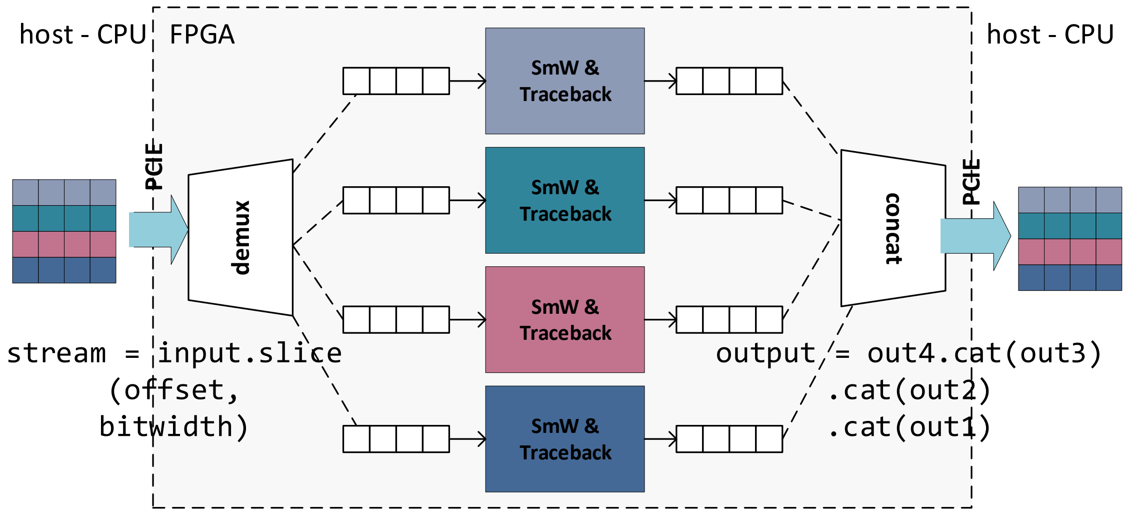


Figure 5.16.: Architecture of Multiple Instances Design.

execution time for Matrix-Fill as well as Matrix-Fill & Traceback when the input dataset ranges from 4K reads to 64 million reads for various number of instances. When doubling the number of parallel instances execution time is reduced by 50% as expected. However, when the clock frequency drops significantly, performance does no longer increase linearly. This occurs for designs with BRAM utilization greater that 80%, which leads to routing congestion to access remote BRAMs and therefore to an increase in critical path delay. For example, in Fig. 5.17a, performance for 4 instances is actually worse than the one for 2 instances, albeit greater than the one of the single instance case.

Multiple Devices

The above design description refers to configuring a single FPGA device. The overall architecture can scale in a straightforward manner by utilizing all devices accessible from a system. The available system is connected to Maxeler's MAX5C platform which incorporates two XilinxVU9P Ultrascale FPGAs. Each of these FPGAs can be configured by multiple instances of the above engines depending on the alignment problem at hand and the need for parallelism. As expected, the execution time is reduced in half when utilizing both FPGAs for a single instance of the full Matrix-Fill and Traceback engine.

Fig. 5.17b represents the scaling of the full design for different configurations of the number of devices and instances leveraged. In terms of performance, the combination of 2 FPGAs running 3 instances of the Matrix-Fill and Traceback module each, is the most favorable one, achieving a ×3.6 speedup, and even outperforms configurations with

Table 5.3.: Resource Utilization and Clock Frequency for Multiple-Instances Architecture.

Configuration		Utilization			Clock
Engine	Instances	BRAM18	DSP	Logic	(MHz)
SmW (Matrix Fill)	1	8.73%	0.04%	6.24%	200
	2	14.49%	0.09%	11.46%	200
	4	26.04%	0.18%	21.7%	200
	8	49.24%	0.35%	42.24%	200
	12	72.99%	0.53%	63.04%	200
SmW & (Matrix-Fill& Traceback)	1	24.75%	0.10%	8.68%	200
	2	45.25%	0.20%	15.70%	200
	3	65.53%	0.31%	22.62%	200
	4	85.79%	0.41%	29.55%	65

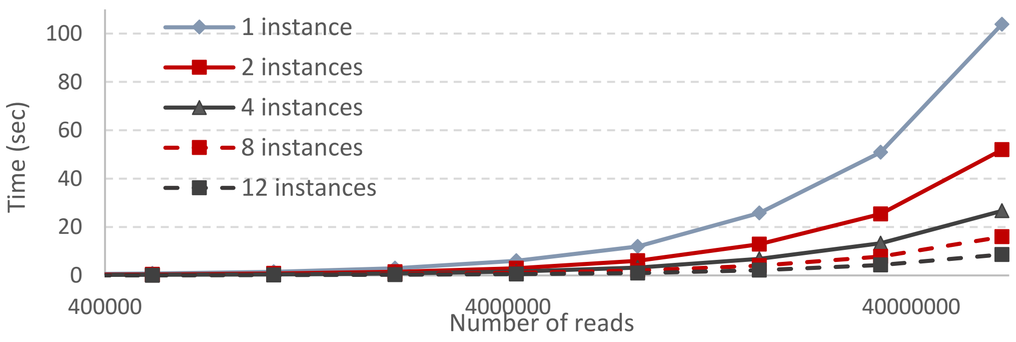
greater total number of instances.

5.5. Accelerator Integration with Bowtie2 Aligner

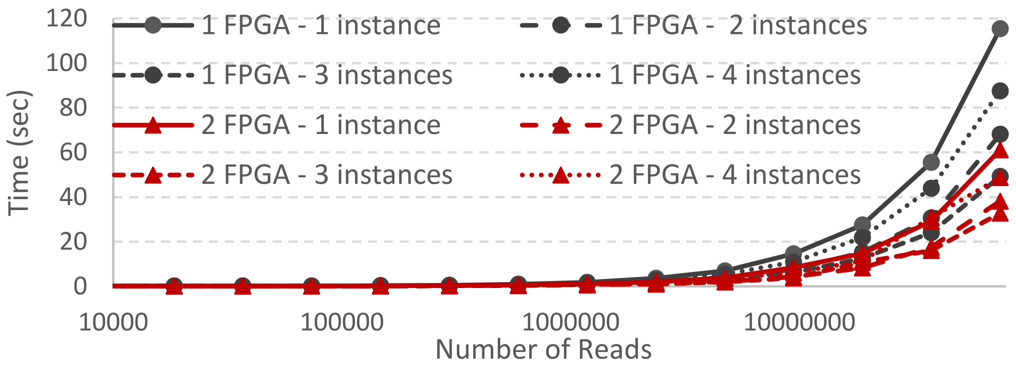
This section describes the integration of the dataflow engines in Bowtie2 aligner. An extensive profiling study of Bowtie2 presents data supporting of the architectural decisions both on hardware and software level. The profiling is followed by a detailed description of the software restructuring that implements the aforementioned decisions.

5.5.1. Alleviating Integration Implications

According to the Bowtie2 algorithm described in the Section 5.3.2 each read invokes a different number of SmithWaterman tasks whose order and number are decided in the runtime. Profiling short read datasets of real patients (CLL) and simulated ones (NEAT) (see more in SectionA.4) provides a detailed overview of the number of tries attempted for each read in the dataset. Fig.5.18a demonstrates that depending on the dataset, Matrix-Fill and Traceback consume up to 48% of total Bowtie2 execution time. Fig.5.18b however highlights that depending on the dataset each read can invoke 1 to 170 tries, i.e. Matrix-Fill tasks. This is equivalent to 500 and 158 million tries in total for the NEAT and CLL dataset respectively. A decision to target only the major bottleneck, i.e. Matrix-Fill and integrate a Matrix-Fill accelerator into Bowtie2 would introduce two major types of overhead. This dictates important architectural decisions for the proposed design that manage to avoid the overheads and allow for hardware acceleration.



(a) MF:Many instances-Single FPGA.

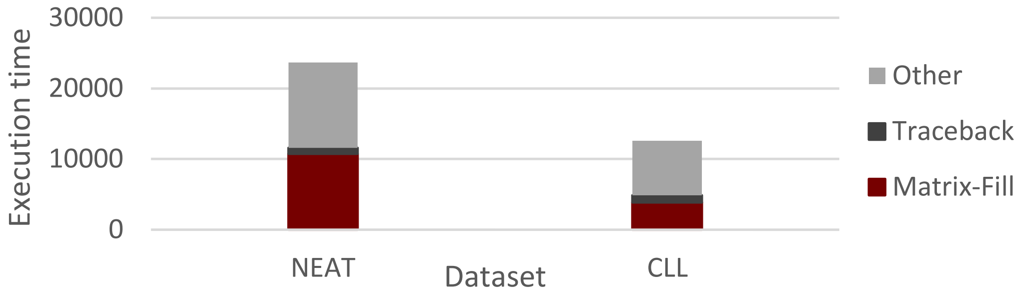


(b) MF&Tr:Many instances-Many FPGA.

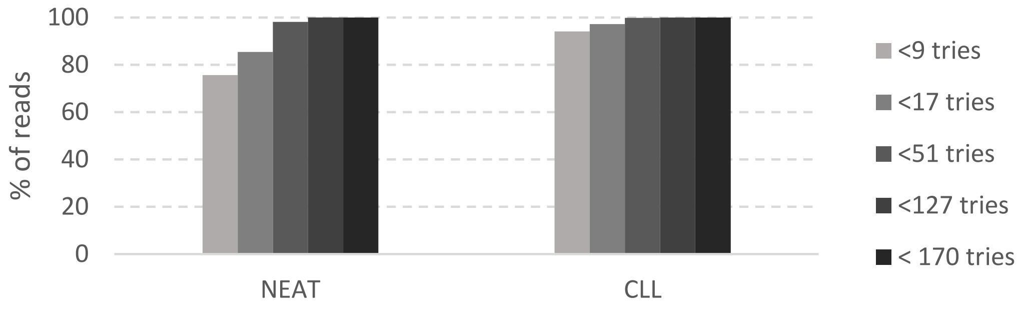
Figure 5.17.: Performance scaling for Various Configurations {#instances, #FPGAs} for dataflow engines for scaling number of reads (4K to 60 million in logarithmic scale), (MF=MatrixFill,Tr=Traceback).

The data transfer overhead is attributed to the cost of sending the matrices E, F, H back to the CPU for Traceback. For CLL and NEAT, this translates to sending back 186 and 54 Terabytes respectively. To eliminate this overhead, the Traceback computation is also implemented on hardware. Therefore, even though the data sent to hardware remains the same, only the final alignment information is sent back to CPU. This adds up to 5Gb of data for both datasets, which corresponds to a compression rate of $\times 38189$ and $\times 11124$ for CLL and NEAT, respectively.

The second overhead is due to the accelerator invoking cost and is dependent on the utilized target platform for acceleration. The technology targeted in this work introduces a 1ms overhead when calling the accelerator. For the immense number of Matrix-Fill tasks reported in the profiling under study, the total overhead actually aggregates to days of execution time for a straightforward integration of accelerator logic. This leads to a prohibitive slowdown, regardless of any speedup acquired from the pure matrix-fill computation. This observation is also supported by [228], which integrates FPGA accelerators for SmithWaterman in Apache Spark. The authors stress that JVM-to-host data copy and host-to-FPGA data transfer aggregates for all SmithWaterman invocations and lead to a slowdown of $\times 1000$. In order to tackle this similar implication, this design proposes a major Bowtie2 software restructuring to constrain the number of acceleration calls and thus avoid the calling overhead.



(a) Breakdown of execution time per task in Bowtie2.



(b) Distribution of tries per read for NEAT and CLL datasets.

Figure 5.18.: Study of Bowtie2 software for different datasets.

5.5.2. Proposed Co-designed Bowtie2

The proposed design rearranges the original Bowtie2 software with the intent of creating batches of candidate alignments, which get aligned on hardware during a single accelerator call. The proposed restructuring splits the original algorithm in three separate phases; a data-gathering phase, a hardware execution phase and lastly a data-distribution phase. Algorithm 2 includes a high level description of the modified algorithm.

In the *data-gathering* phase, instead of running Bowtie2 from start to end for each single read, we iterate over N reads and construct a batch of candidate alignment pairs for all of them (line 3 Alg.2). For each of the N reads, seed extraction and prioritization are performed. As opposed to normal execution, which decides on run-time on the number of tries for seed extension based on alignment results of previous seeds, the proposed design formulates a-priori up to *max_tries* (line 7) candidate pairs and includes them into the accelerator input. Both values N and *max_tries* are configurable. Once the input is constructed, the accelerator is invoked for the *execution phase*, that includes the

Algorithm 2: Bowtie2 Modified

```
1  while not eof in FASTQ do
2      //gathering phase
3      for 1 to N do
4          rd = next_read()
5          seed_list = searchAllSeeds(rd)
6          ranked_seeds = rankSeedHits(seed_list)
7          for 1 to max tries do
8              seed = next_seed(ranked_seeds)
9              (rd,rf) = form_extension(seed)
10             fpga_input = add(rd,rf)
11     //HW execution phase
12     fpga_output = accelerator(fpga_input)
13     //data distribution phase
14     for 1 to N do
15         result_list = distribute(fpga_output)
16         ranked_res = rank(result_list)
17         report(ranked_res[0])
```

matrix-fill and traceback modules running on hardware (line 12). The *data-distribution phase* is executed on software. During this phase, a loop iterates over the output streams to construct a valid alignment result and insert it in a list of all alignments for the given read. Subsequently, the alignments of each read are ranked and the highest-ranking one is reported. Fig. 5.19 illustrates the three-phase restructured Bowtie2, assuming $maxtries = 8$ and interleaving factor $L = 2$. A single acceleration call receives as input $N * maxtries$ candidate read-reference pairs. In order to seamlessly invoke the accelerator and exploit all the parallelism it provides, the construction of the input streams is compliant with the data interleaving technique explained in Section 5.4. Fig.5.19 illustrates that $maxtries$ pairs per read are examined and that the interleaving schema takes place across candidates of the same priority that belong to subsequent reads, i.e. circled candidates. The interleaved groups of reads make up the overall input to the accelerator.

Once the input is constructed, the accelerator is invoked for the *execution phase*, that includes the matrix-fill and traceback modules running on hardware (line 12). The *data-distribution phase* is executed on software. During this phase, a loop iterates over the output streams to assess the result of the alignments and distributes the data to data structures read by I/O Bowtie2 functions. For each seed alignment result, the list of edit operations is traversed in order to construct a valid alignment result and insert it in a list of all alignments for the given read. Subsequently, the alignments of each read are ranked and the optimal is reported.

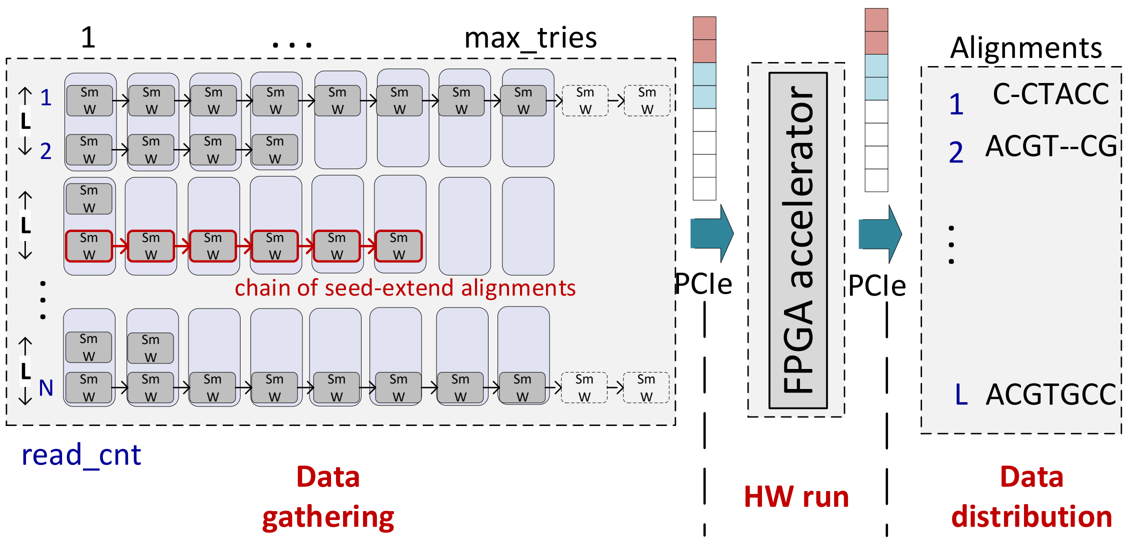


Figure 5.19.: Restructured Bowtie2 Three-phase Algorithm. For simplicity, only a single batch of L reads is illustrated.

The modified algorithm can be configured based on the input dataset to meet both time and accuracy requirements. An exploration for N, $maxtries$ is required to identify their optimal values. A detailed analysis on the accuracy vs. performance sensitivity of max_tries is provided in Section 5.6.3.

Input buffer size sensitivity: The three-stage algorithm is executed iteratively in batches of N until the input reads are exhausted (for loop in Line 3 nested inside outer while loop). The value of N determines both the total number of acceleration calls as well as the amount of data stored required for the data-distribution phase for output reporting. Depending on the size of the input dataset and the memory specifications of the platform, there is a value for N which minimizes the total accelerator call overhead and constraints the memory requirements. An exploration is performed to identify this value for N. For that purpose, 30 million reads are aligned in total for scaling values of N. Fig.5.20 presents the results and indicates that for too small N value the execution severely slows down and can be worse that the software execution utilized by Bowtie2 due to the accelerator overhead call. On the other hand, N does not seem to deteriorate the performance for value greater than 64K reads. Thus, depending on the input dataset size, there is a value for N for which the acceleration call overhead is negligible.

Limit for alignment attempts:The order and number of seed-extension alignments examined within each single read alignment is decided in run-time, causing an inherent irregularity among alignment of different reads. The proposed architecture takes into account this variability and allows for the number of seeds examined per read to be configured depending on the input dataset requirements. This is configured by substituting

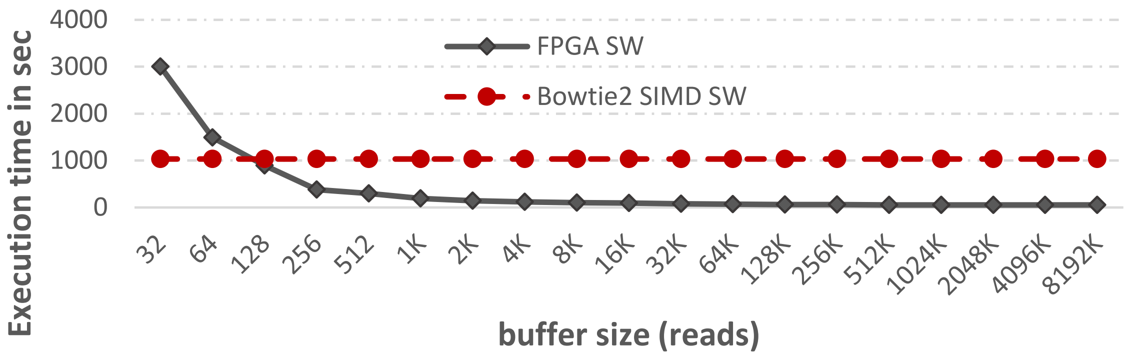


Figure 5.20.: Impact of input buffer size on accelerator-call overhead and thus execution latency.

the *while* loop in the original algorithm with a *for* loop with fixed upper limit in line 7 of the modified algorithm. The quality of the input reads defines the value of the upper limits so that the alignment accuracy is preserved. An evaluation of this parameter for different datasets is provided in Section A.4.

5.6. Experimental Results

5.6.1. Experimental Setup

We evaluated GANDAFL on a high-end architectural prototype consisting of a Maxeler's MAX5C DFE (dataflow engine) with Xilinx VU9P Ultrascale FPGAs, integrated through PCIe with a dual socket Intel Xeon E5-2658A (v3) CPU operating at 2.2GHz with 128GB DDR4 DRAM clocked at 2133MHz. Each DFE consists of a large capacity arithmetic chip, x8 PCIe Gen 2 connectivity and 38 MB on-chip SRAM. GANDAFL was implemented using the MaxJ dataflow computing model [239]. MaxCompiler version 2018.1 and Vivado 2017.4 were used for synthesis. Bowtie2 version 2.2.3 is invoked through SeqMule [240] for automated human genome/exome variant detection.

All experiments perform alignment against GRCH38 (hg38) human genome assembly, indexed by Bowtie2 software and are executed on real hardware. For testing, we create three different datasets of length 50, 100 and 150bp based on popular NEAT simulator [241], which inserts errors and mutations based on a sequencing error model. We selected $\times 20$ coverage and error rate 0.01% to match the trend in modern sequencers for short-reads [242], [238]. We also leverage CLL dataset, a 60 million short-read input dataset of read length 100bp with coverage $\times 30$ and error rate 0.01%, collected as part of EU healthcare project AEGLE [243] for research for Chronic Lymphocytic Leukemia [244].

5.6.2. Accelerator Evaluation

At first we evaluate the efficiency of GANDAFL as a standalone component. This type of alignment acceleration is essential for researchers to perform various tasks, such as species identification or comparison based on a single gene. In favor of fast execution, researchers do not employ SmithWaterman but rely on algorithms (such as BLAST [117]) which trade some hard-to-find hits for speed. However, a hardware accelerated SmithWaterman, that guarantees better accuracy, would show great potential.

We consider a comparative study of GANDAFL with respect to other software and hardware high performance implementations: i) Edlib [114], a C/C++ library for fast, exact sequence alignment using edit distance, (ii) WFA [115], a wavefront alignment for exact gap-affine pairwise alignment, (iii) KSW2 [116], a C library to align a pair of biological sequences based on dynamic programming, (iv) the SSE-vectorized heuristic SmW software implementation utilized in the SIMD optimized Bowtie2 [113], (v) GenASM (software version) [136], an approximate string matching (ASM) acceleration framework for genome sequence analysis, vi) the open-source SmW dataflow implementation available on Maxeler AppGallery [235] and vii) the state-of-the-art open-source Darwin sequencing accelerator [36]. The Maxeler Dataflow accelerator does not include the traceback stage, but simply produces the maximum score and the respective position for each alignment. It includes 512 PEs and operates on 150MHz. Darwin includes and provides an open-source RTL accelerator, i.e. GACT, that implements the Matrix-Fill & Traceback operations and can be synthesized for 250MHz. The GACT array in Darwin was simulated for 128 PEs, since it achieved slightly better results than the 64-PEs optimal architecture reported in the respective paper. For GANDAFL, the achieved frequency is 200MHz.

Fig.5.21a shows the comparative results in terms of accelerator throughput (aligns/sec) for three NEAT datasets of dominant/representative read lengths [**?**] utilized in NGS analysis. This comparison utilizes a single-instance of the SmithWaterman accelerator for all designs and targets Xilinx VU9P Ultrascale FPGA board for the FPGA accelerated ones. All software implementations are executed on an Intel Xeon Gold 5218 which is based on 14nm technology to match the technology of the targeted Xilinx VU9P Ultrascale FPGA. As shown in Fig.5.21a, GANDAFL outperforms WFA and software GenASM by two orders of magnitude, KSW2 by $\times 59$, Bowtie2 by $\times 98.42$ and Edlib by $\times 22$ for 100bp reads. Edlib is in fact the fastest software aligner. GANDAFL outperforms Maxeler dataflow by $\times 9.5$ and Darwin GACT by $\times 2.13$. The throughput for all designs follows a downward trend when the length of the read sequences increases, even by 50bp. However, the accelerators preserve their relative throughput. GANDAFL retains its speedup over state-of-the-art Darwin accelerator, showing in fact a marginal increase for increased read length.

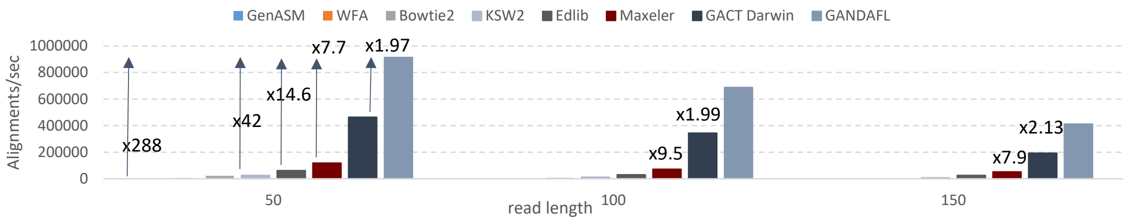
Chip-to-chip: In order to evaluate how efficiently each design exploits the underlying resources, we perform a chip-to-chip comparison for hardware accelerators, i.e. we evaluate the performance of the examined accelerator designs under the scenario that each one configures the entire FPGA with as many instances as possible, thus enabling parallel acceleration through maximal resource utilization. For all designs we target the same board (Xilinx VU9P Ultrascale), consider the same PCIe 3.0 interface. We perform an iterative exploration for each design up to bitstream generation (synthesis and place route) to find the optimal configuration in terms of #ParallelAccelerators $\times$ F_{clk}. We then evaluate the throughput achieved by each design for a real-world scenario that aligns the CLL dataset of 100bp long reads. For each many-accelerator design, we account for both processing as well as data transfers. The critical resource for all designs is the BRAM. We examine configurations that utilize up to 80-85% of the available BRAMs, as these were able to meet timing constraints. For GANDAFL, we utilize the second to last configuration of Table 5.3, as it represents the maximum instances with high frequency, i.e. 3 instances clocked at 200MHz. The Maxeler accelerator can fit $\times$14 onto the FPGA while preserving its F_{clk} to 150MHz. For the GACT accelerator, we leverage a 128-PE design that fits $\times$25 on the FPGA and operates at 150MHz. The comparative throughput results are presented in the Fig.5.21b As shown, in this chip-to-chip comparative study, the proposed solution outperforms both Maxeler and GATC many-accelerators. In comparison to the GACT, GANDAFL delivers a $\times$5.5 gain in throughput, showing the effectiveness of dataflow design. While GANDAFL overlaps the data transfer with the computation, GACT operates on batches of 25 reads per accelerator invocation, thus adding a significant data transfer overhead. In comparison with the Maxeler many-accelerator design, GANDAFL delivers a $\times$8.99 gain in achieved throughput showing the effectiveness of the employed optimized dataflow implementation, which enables deeper and higher clock frequency pipeline with higher utilization efficiency due to extensive interleaving.

5.6.3. Integrated Architecture Evaluation

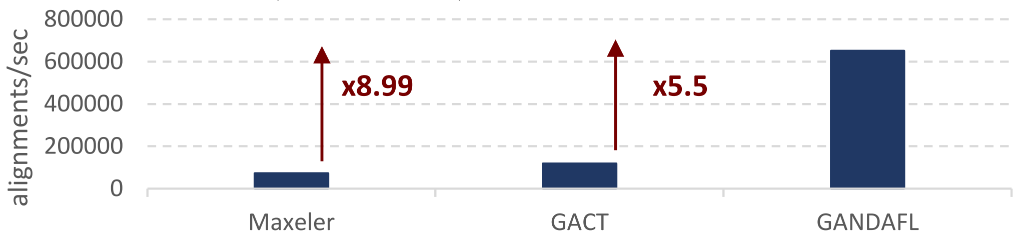
GANDAFL is also evaluated as part of an end-to-end aligner, that receives input reads in FASTQ format and produces the output in BAM format. The end-to-end aligner is evaluated both in terms of accuracy and performance. We utilize the CLL and NEAT 100-bases long datasets to evaluate the impact of the quality of reads on these metrics.

Accuracy Loss Study: Biomedical applications introduce strict accuracy constraints. In order to ensure the correctness of our design, we performed validation of all components and intermediate results. Given a read-reference candidate pair to align, the dataflow accelerator is guaranteed to deliver exactly the same results as Bowtie2. In the integrated version with Bowtie2, the proposed solution relaxes the constraints regarding the number of seed extensions examined when aligning a read (Section 5.5.2).

In order to evaluate the potential impact of this relaxation, we performed an extensive



(a) Throughput comparison for different read lengths among GenASM, Edlib, WFA, KSW2, Bowtie2 SmW, Maxeler(only MatrixFill), GACT Darwin, GANDAFL.



(b) Chip-to-chip comparison of throughput for hardware aligners Maxeler(MatrixFill), GACT and GANDAFL.

Figure 5.21.: Throughput evaluation of aligners.

profiling and verification study on the NEAT and CLL datasets. The above analysis, as depicted in Fig.5.18b shows that 95% of the reads of CLL dataset and 75% of the reads of the NEAT dataset are aligned within examination of 8 seed-extension candidate pairs. In both datasets, 99% of the reads require up to 50 tries whereas a negligible number of reads can examine up to 160 candidates. It is also worth mentioning that constraining the number of tries per read does not necessarily decrease the accuracy, as the candidate seed read-reference pairs are not randomly examined. Bowtie2 ranks them so that the most probable matches are tried first. Fig.5.22 shows that the read-reference candidate that delivers the reported alignment is in fact the first one examined for 94.31% and 86.92% of the reads for NEAT and CLL datasets respectively. Furthermore, if an alignment has not been found within the first tries, it is less probable that this read aligns at all. Table 5.4 includes the alignment rates for the two datasets when inflicting an upper limit of 8 for the tries per read. The loss in successful alignments is less than 1.2%. Furthermore, a small percentage of reads, ranging from 0.83 to 2%, is included in the reads that align only once instead of multiple times. However, the reported alignment in these cases is the same.

Depending on the dataset, constraining the number of tries per read does not greatly impact the accuracy, but can yield time savings. An analysis has been conducted to quantify the trade-off between total alignment success rate and performance degradation. Fig. 5.23 illustrates that the accuracy increases slower for greater number of checks as opposed to the increase in the execution time. Therefore although our design sacrifices accuracy, it exhibits great resilience and tolerance to significant accuracy

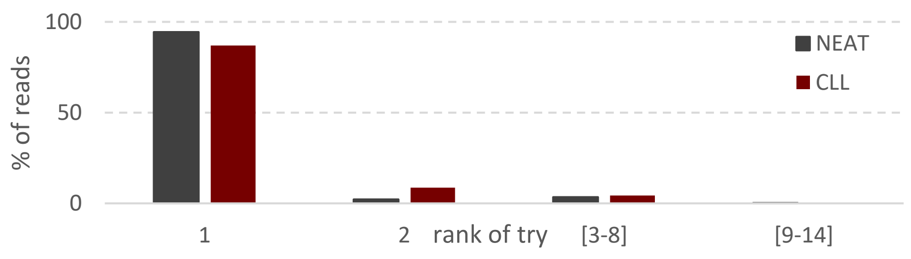


Figure 5.22.: Distribution of the rank of tries that delivers the reported alignment across reads.

Table 5.4.: Accuracy comparison Bowtie2 vs Proposed.

	Bowtie2			Proposed		
times aligned	0	1	>1	0	1	>1
NEAT	0.36%	87.48%	12.16%	1.62%	89.5%	8.88%
		99.64%			**98.38%**	
CLL	0.73%	69.70%	29.57%	1.32%	70.53%	28.15%
		99.27%			**98.68%**	

losses.

Assessing the Number of Edits Constraint: The accuracy of the results can also be affected from the number of edits identified by our design for a single alignment. The minimal loss in accuracy of our design is also attributed to efficient handling of edit constraints. Our design is fully parametric and can be configured to support any number of edits during compile time, i.e. we statically allocate memory resources in order to store any potential edits and send them back to CPU for alignment reconstruction and reporting purposes. Therefore, the number of edits explored is a design decision rather than a design limitation.

With that being said, it would be of value to examine if such a limit in the number of edits can support alignment in real-world problems. Elaborate studies[1],[2] show that 2nd and 3rd Generation sequencing platforms produce short reads with error rate less than 1%. This means that most occurrences in alignment are matches and a margin of 17 edits for up to 150 base-long reads is sufficient. From a practical point of view too, our hardware implementation cannot reconfigure the size of edits memory on runtime. We have to set an upper limit for the expected edits per alignment. Extensive profiling was conducted for the available datasets to pinpoint an upper limit without greatly

[1]Glenn TC. Field guide to next-generation DNA sequencers. Mol Ecol Resour. 2011 Sep;11(5):759-69. doi: 10.1111/j.1755-0998.2011.03024.x. Epub 2011 May 19. PMID: 21592312

[2]Ross, M.G., Russ, C., Costello, M. et al. Characterizing and measuring bias in sequence data. Genome Biol 14, R51 (2013). https://doi.org/10.1186/gb-2013-14-5-r51

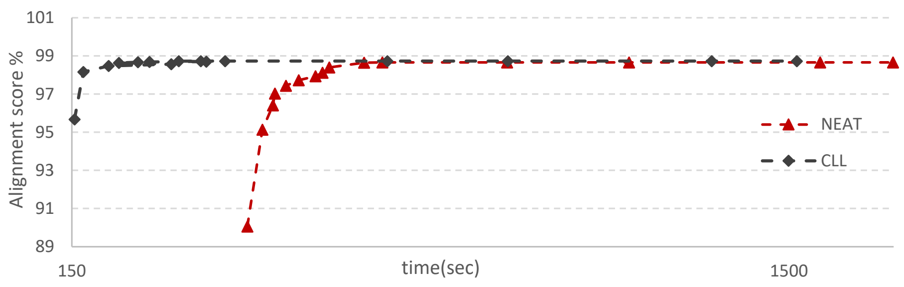
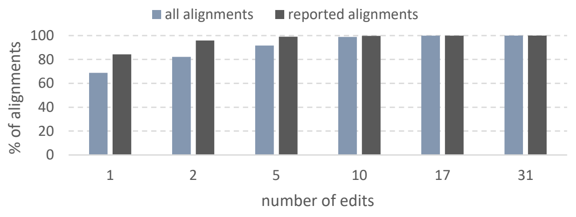
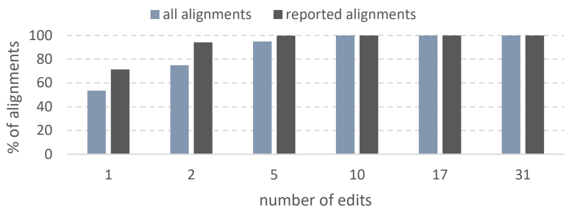


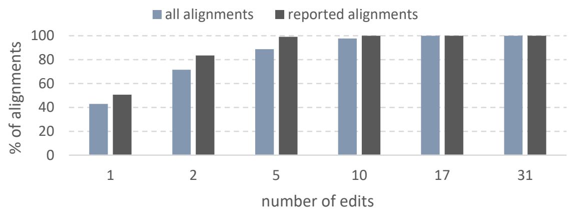
Figure 5.23.: Tradeoff of Alignment success rate and performance for scaling number of candidates per read.



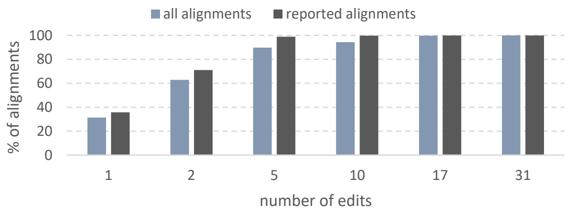
(a) CLL dataset of length 100



(b) NEAT 50-length dataset



(c) NEAT 100-length dataset



(d) NEAT 150-length dataset

Figure 5.24.: caption

sacrificing accuracy. Fig.5.24 presents an histogram of the number of edits for all alignments found for each dataset. This is also compared with the equivalent histogram that only takes into account the edits found in alignments that get reported. As a general trend, the reported alignments include fewer edits. For all datasets, the 99.99% of alignments contain fewer than 18 edits. Therefore, we decide to use this upper limit for our design.

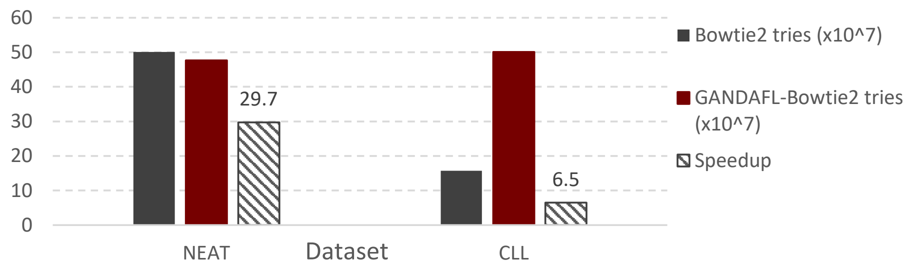
The works in relevant literature apply similar constraints in the number of edits examined. Maxeler SmW accelerator in particular, only implements the Matrix Fill stage on hardware and performs the traceback on the CPU, therefore the number of edits are not relevant in this implementation. GACT accelerator on the other hand handles long reads, which exhibit greater error rates by definition. The datasets constructed in this case have error rates that range from 15 to 40%. Taking into account this fact, the blocking scheme implemented and the lack of special reference to the topic, it is safe to deduce that GACT does not limit the number of edits discovered. However, this is not a realistic scenario in short read alignment, as presented by the trend in error profiles in [238], [242]. A striking example of this fact is that ASAP architecture relies on the small error rate (<1%) of short reads to boost the performance of the alignment. In particular, it is assumed that most comparison between bases are matches and they correspond to zero-delay. Increasing the number of mismatches, increases the latency of the alignment. Although a limit for the number of edits is not provided, the authors utilize a representative paradigm of their architecture with a maximum edit distance (i.e. number of edits) of 6 and mention that the maximum tolerable edit distance is based on the application.

Single-thread Integration efficiency: We evaluate the performance enhancement, when integrating a single instance of GANDAFL of 100 PEs, clocked at 200MHz in a single-thread Bowtie2 examining 8 candidates/read.

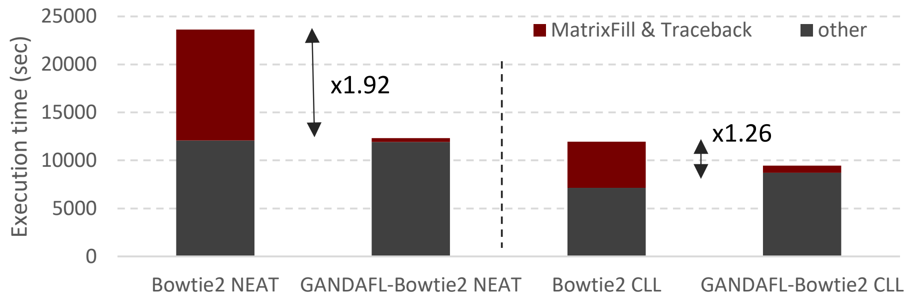
Fig.5.25 presents the acquired speedup considering CLL and NEAT datasets. Fig.5.25a focuses only on the SmithWaterman operations that are accelerated on hardware. For each dataset, the first two bars stand for the number of Matrix-Fill&Traceback tasks examined by Bowtie2 and the proposed design, respectively. For the NEAT dataset the proposed design examines less candidates, i.e. the ratio is 0.95, whereas for the CLL dataset the respective ratio is 3. The number of candidates tried appears to be correlated with the quality of the generated reads. In this case, the CLL dataset was meticulously generated by experts, which led to a dataset of high precision and low error rate. This in turn translates to fewer matching positions in the genome. Specifically, as reported in Fig.5.18b, 95% of the reads invoke less than 9 tries and in fact 90% of the reads require only 3 tries. This leads to the proposed design performing $\times 3$ more tasks. Nevertheless, there is still a speedup of $\times 6.5$. In the case of the NEAT dataset, which was generated with a moderate coverage percentage and higher error model, the number of candidates is similar to the pure software Bowtie2, leading to a more impressive speedup of $\times 30$.

Fig.5.25b depicts the impact of the latter speedup to end-to-end performance. For the NEAT dataset, the results are very close to the Amdhal's law optimal speedup. The SmithWaterman operations take up 49% of the total execution latency in the original Bowtie2. Even if the hardware acceleration annihilates this time, the maximum speedup that can be achieved is $\times 1.96$. The proposed design achieves $\times 1.92$ speedup. The deviation from the optimal speedup is greater for the CLL dataset. In this case, SmithWaterman takes up 40% of the execution time of Bowtie2, which corresponds to an upper limit of $\times 1.67$ speedup. The proposed design manages to deliver a $\times 1.26$ speedup. This is attributed to a 22% increase in the pure software execution time, due to handling $\times 3$ more data than the original Bowtie2. Therefore, the performance gains are dependent on the initial bottleneck and the quality of the read dataset.

Integration efficiency in an accelerator sharing scenario: In order to exploit the full capabilities of our available computing resources and perform a fair comparison between a SW-



(a) Comparison between original and GANDAFL-Bowtie2.



(b) End-to-end speedup of GANDAFL-Bowtie2.

Figure 5.25.: Performance of Bowtie2 integrated accelerator with $max\_tries = 8$.

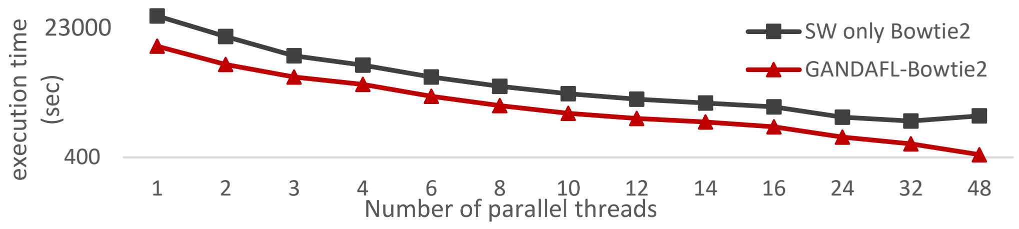


Figure 5.26.: Comparison performance for scaling number of threads while sharing two accelerators.

only and the proposed SW/HW accelerated Bowtie2 (GANDAFL-Bowtie2), we perform a study that leverages all available software threads and FPGAs through accelerator sharing. In particular, we examine how the proposed accelerated Bowtie2 leveraging the two FPGA devices scales its performance when we increase the number of software threads. Since the #threads > #FPGAs, the deployed FPGA accelerators are shared. Therefore, we examine how efficient the accelerator sharing can be among multiple threads and if it can boost the overall performance w.r.t. the pure software multi-threaded Bowtie2 deployments.

Each FPGA is configured with a single instance of our accelerator and a First-Come-First-Served (FCFS) allocation/sharing mechanism is implemented. The results in Fig.5.26 show that sharing the two FPGA accelerators among multiple threads is always more efficient than pure software multi-threaded execution, delivering speedups ranging between

$\times 1.73$ up to $\times 3$ for 48 threads. The efficiency of the accelerator sharing is attributed to the scheduling of consecutive smaller alignment tasks rather than a single immense one that balances and overlaps the wait time across threads. As a result, multiple threads can lock the FPGA and any potential wait time is not prohibitive.

Chapter 6.

Profile-Driven Banded Smith-Waterman acceleration for Short Read Alignment

Short read alignment is a critical step in genomic pipelines that requires optimization due to the enormous input size and complexity of SmithWaterman string matching. Several optimization techniques have been examined, such as hardware acceleration, heuristics and pre-filtering. This work combines these approaches into a single powerful solution that leverages the low edit rate of reads and the principles of Banded SmithWaterman, to highlight the value of creating accelerators customized to the input accuracy requirements. Extensive profiling of genomic datasets reveals low edit thresholds that can be leveraged by Banded SmithWaterman to create resource-efficient accelerators that are customized to the edit profile of the input. We first design and deliver a highly optimized dataflow FPGA-based implementation for Banded Smith-Waterman seed-extension, which can perform alignments given an upper edit threshold. A multi-dataflow system is then configured with multiple Banded accelerators covering the full range of edits to achieve both high throughput as well as high accuracy alignment. The result is a dataset-specific multi-dataflow design that leverages pre-filtering to guide the alignments to the suitable Banded SmithWaterman instance and meets the demands of the datasets in both throughput and accuracy. This work was accepted for publication in the Design Automation Conference 2023.

6.1. Introduction

Genomics is a fast-evolving research domain, that allows scientists to understand the mechanisms responsible for the genetic diversity. The growth in the field has been facilitated by advances in sequencing technologies, that have brought about the generation of billions of short fragments of DNA at low cost and increased quality, with error rates as low as 0.01% [242]. The nucleotide short *reads* generated by sequencing platforms are utilized to reconstruct the sample genome and compare it to the reference genome as part of various analyses such as variant calling and epigenetics [245]. Such analyses reveal differences between the genomes, attributed either to sequencing errors or inherent genetic variations. The genome reconstruction is achieved through *short read alignment*, that maps the short reads to a location in the reference genome that is most likely its origin. Most aligners [112, 113, 117] adopt a *seed-and-extend* strategy to find possible matches of the read on the reference genome. *Seeding* fragments each read into even shorter pieces called *seeds* that align exactly on the reference genome and creates a pool of candidates for valid alignments. In the *seed extension*, each seed is extended into a gapped alignment, i.e. allowing mismatches or *edits*.

In modern aligners, the extension step is most frequently implemented based on Smith-Waterman [118] dynamic programming algorithm for string matching that operates in two stages. Matrix Fill fills a similarity score matrix between a read and reference sequence. The fewer the gaps and mismatches between the sequences, the higher the score of each alignment is and therefore the similarity of the two sequences. Traceback stage starts from the cell in the similarity matrix with the highest score, and traverses the matrix backwards until it reaches a zero-score cell, i.e. the first match point of the two sequences. During backtracing, it identifies all potential edits and thus reconstructs the alignment path. The excessive time requirements and computational intensity of this step however forms a major bottleneck and the need for optimization is imperative. As mentioned in [173], performance optimization efforts follow two different approaches: i) the first one targets optimization of a single alignment task [36, 123], whereas ii) the alternative focus on decreasing the volume of alignment tasks [119].

Hardware acceleration of alignment tasks: The first category can be further distinguished into solutions that leverage hardware acceleration and solutions that employ heuristic techniques or a combination of the two. Hardware accelerations for SmithWaterman have been developed for a variety of devices such as GPUs, ASICs and FPGAs. Due to their bit-level customization capabilities, FPGAs have emerged as promising Smith-Waterman accelerators both for industry [26] and academia [36, 246]. Most of them, [123], [124] are based on a wavefront approach that implements a pipeline of PEs as a systolic array. The majority of them focus on providing a highly optimized accelerator for the first stage, however they do not provide a Traceback implementation, which excludes them from integration into commonly-used software aligners. Edlib [114] software aligner implements the alignment by using vectorized operations and replacing SmithWaterman with a simpler algorithm that calculates the Levehnstein distance instead. Other software aligners rely mainly on heuristic techniques to improve performance. For example, Bowtie2 [113] utilizes a heuristic of SmithWaterman that ignores selected dependencies in order to leverage SIMD instructions and adds a correction step to account for any errors. Another common heuristic is Banded SmithWaterman [247], which only looks for alignments that adhere to an edit threshold, i.e. around the diagonal of the matrix. Recent works target Banded SmithWaterman for accelerating either long reads alignment [248] or sequencing datasets of high sequencing error rates [249].

Seed-extension pre-filtering: The second category is represented by pre-filtering algorithms that aim at decreasing the vast amount of seed extension tasks, by discarding candidate alignments based on a predefined threshold for the edit distance. These candidates would most likely result in a prohibitive number of edits and would be eliminated. Several pre-filtering algorithms [90, 95, 119] have now been developed that have noted significant improvement in accuracy and performance since early efforts. SneakySnake [119] is a state-of-the-art pre-filtering algorithm, as it achieves higher accuracy alignment results through a highly efficient decision technique to discard unnecessary extensions.

Despite the promising results on genomics efficiency, the above research categories are still evolving in isolation, thus limiting the potential gains of a cooperative optimization approach. This is attributed to the fundamentally different scope of each category. On one hand, *genomic hardware acceleration* is driven by *data-agnostic* decisions that built accelerators able to perform accurate seed extension on generic datasets. On the other hand, *pre-filtering optimization* is a fully *data-aware* procedure that delivers best results when customized to the underlying dataset.

In this paper, we bridge hardware acceleration and pre-filtering for alignment optimization by introducing a profile-driven Smith-Waterman accelerator design. We leverage the edit profile of genomic datasets to pinpoint dominant edit thresholds in the alignments and exploit them to configure different Banded SmithWaterman accelerators. The adoption of these profile-driven upper limits and resource-efficient Banded SmithWaterman accelerators enables the provision of a highly parallel Banded Smith-Waterman accelerated system. We design an heterogeneous system, with different accelerators supporting the full range of edit thresholds and employ a pre-filtering algorithm to guide candidate alignments to an accelerator that supports the expected edit threshold, thus delivering a high throughput alignment system without compromising accuracy.

Our main innovations are summarized as follows: (i) we introduce a profile-driven design methodology leveraging Banded Smith-Waterman for seed-extension acceleration customized to the input edit threshold distribution, (ii) we design and deliver a highly optimized dataflow implementation for Banded Smith-Waterman seed-extension targeting FPGA devices and (iii) we implement a dataset-specific multi-dataflow system that significantly accelerates pre-filtering seed-extension alignment with negligible accuracy loss. Through an extensive experimental campaign, we evaluate the efficiency of both the newly introduced Banded Smith-Waterman accelerator as well as the delivered profile-driven multi-dataflow system against a rich set of state-of-the-art alignment solutions. The evaluation shows that the proposed Banded Smith-Waterman accelerator delivers a $\times 34$ speedup over state-of-the-art software aligner and $\times 1.53$ over state-of-the-art dataflow SmithWaterman accelerator [246] and $\times 3$ over GACT RTL accelerator [36]. Moreover, the proposed multi-dataflow system delivers average speedups of $\times 1.8$ over state-of-art multi-accelerator FPGA solutions [246] that employ generic and input-agnostic accelerators, further validating the efficiency of the proposed profile-driven approach for accelerators' parallelism customization.

The rest of the paper is organized as follows: we first motivate and sketch the basic concepts of the proposed profile-driven customization methodology in Section 6.2. In Section 6.3, we design and detail the micro-architectural decisions of the introduced multi-dataflow accelerared system. Section A.4 provides a thorough and in-depth evaluation and comparative analysis of the proposed solution w.r.t. state-of-the-art.

6.2. Profile-driven Genomic Architecture Optimization

In this work, we propose to leverage the edit profile of modern short read datasets in order to create a high-throughput accelerated system fit to the accuracy needs of the input.

Profiling insights: In order to evaluate the impact of the edit profile on the architecture, we meticulously study the behavior and results during alignment of three different input datasets: (i) a simulated dataset of 60 million 100-base long reads created by NEAT [241] simulator, (ii) 60 million reads of often-used sample NA12878 (ERR194147_1) provided by 1000 Genomes Project Release3 [250] and (iii) CLL, a real-patient dataset assembled as part of research for Chronic Lymphocytic Leukemia. Fig.6.1 presents a histogram of the number of edits for all alignments found for each dataset. The selected edit thresholds in the histograms, i.e. 2, 5, 10, 18, 30 edits, correspond to critical values for which a significant increase in the number of aligned reads occurs. For the simulated dataset, 70.8% of the examined aligments include at most a single edit. The same percentage is equal to 74.4% and 82% for the NA12878 and CLL dataset. The number of reads that align for a given edit threshold, increases rapidly (by 5-10%) for threshold ranging from 1 to 8. However, for values greater than 10 the incremental changes is in the order of 0.01%. In fact, for all datasets the 99.99% of alignments contain fewer than 18 edits.

Banded Smith-Waterman: The first valuable insight of the profiling is the potential impact

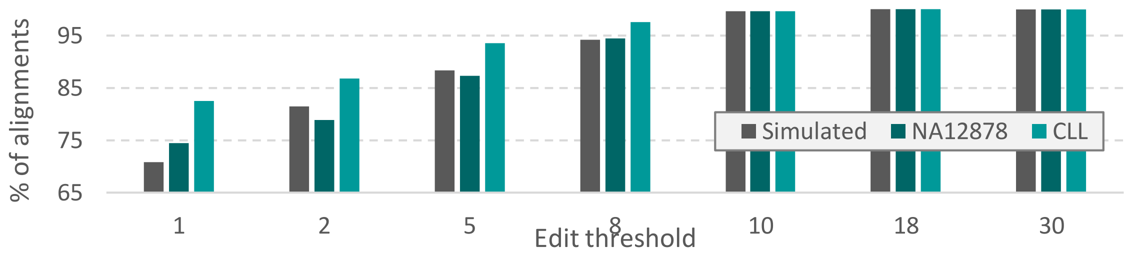


Figure 6.1.: Distribution of number of edits for alignments for three different datasets.

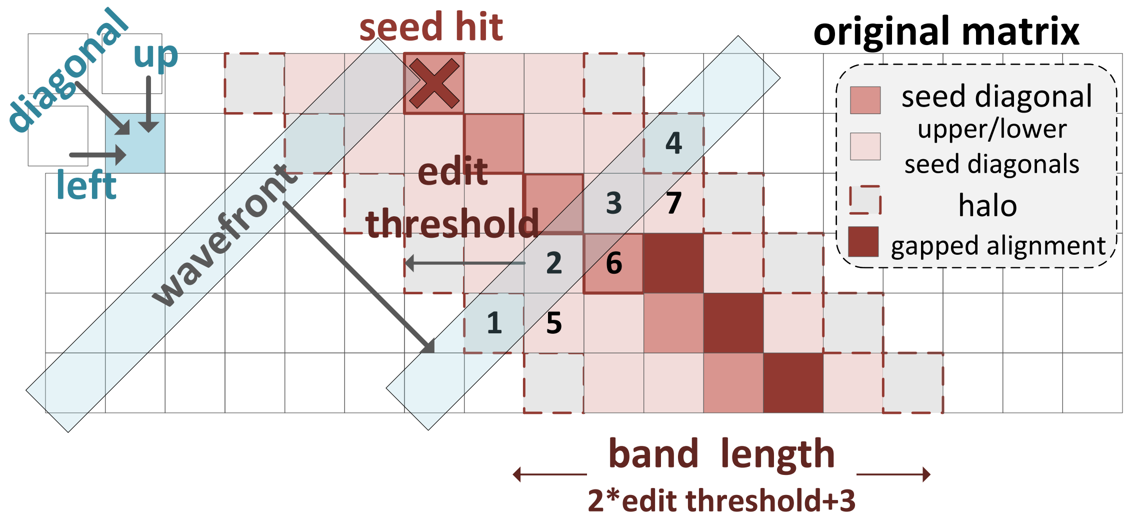


Figure 6.2.: Banded SmithWaterman example for *edit threshold 2*.

of the low edit number on the accelerator architecture. Fig.6.2 illustrates the placement of an exact as well as a gapped alignment on the score matrix. Both alignments start from the seed hit, i.e. the first match point, however the gapped alignment deviates from the *seed diagonal* as it includes an edit. The smaller the edit distance, the narrower the band of the matrix that the alignment spans. This is by definition exploited by Banded Smith-Waterman [247], that is based on the observation that the max score in the matrix appears around the seed diagonal at a maximum distance equal to the number of edits. Therefore, for a given edit threshold, Banded Smith-Waterman focuses on the respective band within the matrix and eliminates the need for storing and searching the entire score matrices. Fig.6.2 demonstrates the band of interest within the complete matrix, after taking the *edit threshold* into consideration. The final alignment will be found within the area marked by the seed diagonal, the *edit threshold* upper seed diagonals and the *edit threshold* lower seed diagonals. As Traceback also checks the neighboring cells forming the *halo* area of Fig.6.2, the band of interest has *band_length* equal to $2 * edit\_threshold + 3$. Therefore, studying the edit profile of the input dataset can help us avoid overprovisioning of resources and design more resource efficient seed-extension accelerators.

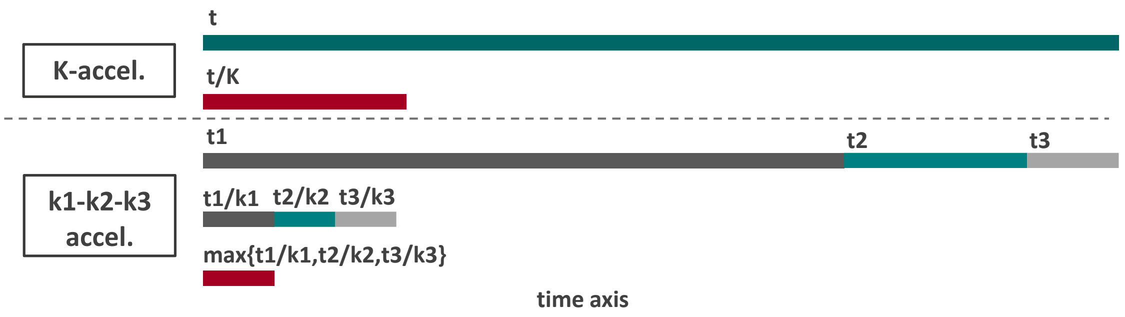


Figure 6.3.: Calculation of execution time when applying K generic accelerators as opposed to applying k_1, k_2, k_3 smaller accelerators fit to an edit distribution of $70 - 20 - 10$.

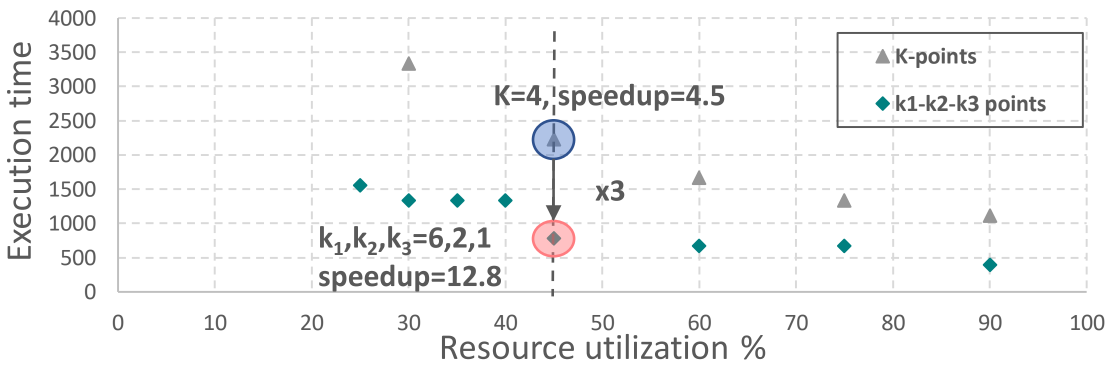


Figure 6.4.: Example of applying generic accelerators as opposed to employing smaller accelerators fit to the input edit profile.

Profile-driven genomic accelerator architecture optimization: The second important insight does not concern the low edit threshold itself but rather the frequency in which each threshold occurs. The distribution per edit threshold straightforwardly translates into a similar distribution of the time taken up by each type of alignment, i.e. alignments with 0-1 edits take up at least 70% of the total dataset alignment time. This heterogeneity suggests that the overall performance could benefit more if we instantiated multiple smaller-sized accelerators and assigned them to the corresponding alignment types in a similar distribution.

Let as assume a system of K generic parallel accelerators that take up the acceleration of the alignment of a given dataset. We claim that there is a configuration of $k_1, k_2..k_l$ edit-customized accelerators that 1) follow the edit distribution of the dataset and 2) satisfy the inequality $\sum_{i=1}^{l} k_i > K$, so that they achieve a greater overall speedup than K for equal resources utilization.

Let us assume a realistic scenario with an edit threshold distribution 70-20-10% for corresponding thresholds of 1, 5, 15 edits, respectively. We also assume Banded SW accelerators i) with area occupation equal to a fraction of the area of the single general accelerator , and ii) with equal execution latencies for all band sizes, e.g. 1.5× faster than the software. Based on this assumptions, we formulate a model that calculates the execution time of the system for a compact search space of different configurations for K, k_1, k_2, k_3. Fig.6.3 demonstrates the calculation of the total execution time. If we take into account the edit distribution and the Amdahl Law, the total time is distributed to each threshold category proportionally to the edit distribution. Then, in the generic case K accelerators accelerate time t, whereas in the proposed strategy k_1, k_2, k_3 accelerate time periods t_1, t_2, t_3 respectively.

Under the above assumptions, Fig.6.4 depicts exploration results for different $K, k1, k2, k3$ accelerator allocations, in terms of latency and resources utilization. As shown, configuration points of k_1, k_2, k_3 fluctuate lower than the corresponding K points for the same resource utilization percentage. For example, $K = 4$ generic accelerators take up 45% of hardware resources and achieve a speedup of ×4.5, where $k_1, k_2, k_3 = 6, 2, 1$ customized accelerators take up as much hardware resources for a speedup of ×12.8.

We leverage the above insights to create a system with multiple dataflow accelerators, i.e. multi-dataflow system, customized to the dataset specifications in terms of edit threshold. As shown in Fig.6.5 this can be achieved by first profiling the input dataset to extract the edit distribution and pinpoint the edit thresholds for which there is a spike in the number of aligned reads. The number of thresholds indicates the number of different types of edit-customized Banded Smith-Waterman accelerators. Each type can support up to a predefined number of edits, equal to the found thresholds. The number of accelerators allocated for each type follows the same distribution as the edit thresholds. Followingly, SneakySnake [119] pre-filtering algorithm classifies input candidate alignments into the respective edit threshold category and assigns each alignment to the corresponding accelerator.

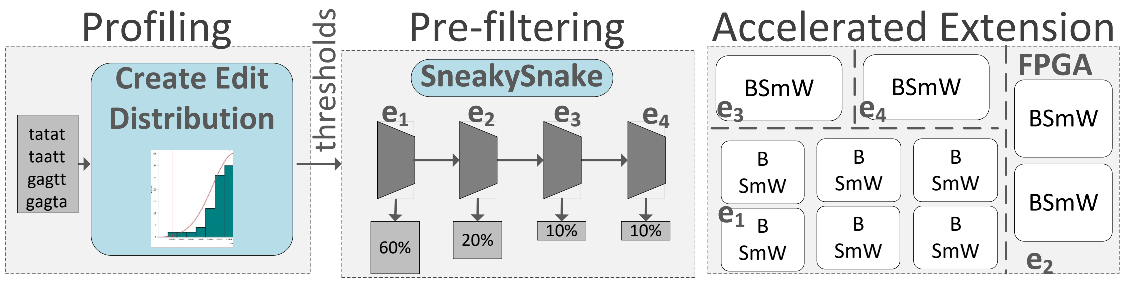


Figure 6.5.: Overview of profile-driven acceleration strategy and system architecture for a distribution of 60-20-10-10%.

6.3. Design of Dataflow Genomic Accelerator

The proposed multi-dataflow design includes multiple single Banded Smith-Waterman accelerators, which have been developed utilizing dataflow computing. Each one is configured with an edit threshold selected at profiling based on the proposed methodology. The single Banded Smith-Waterman accelerator includes two computing units, one for each of SmithWaterman stages, i.e. Matrix-Fill and Traceback. Matrix Fill receives read-reference pairs as alignment candidates and computes the matrices E,F,H. Once the matrices are ready, they are traversed in reverse order by Traceback to identify edits. At any time, they execute in parallel and in pipeline manner, i.e. as Matrix Fill operates on an alignment pair, Traceback processes the matrices created for the previous pair.

Matrix Fill Banded Design: Fig.6.2 ilustrates the dependencies and parallelism of Banded Smith-Waterman. The computation of matrices E,F,H by Matrix Fill is characterized by anti-diagonal dependencies, as each cell value depends on the left, up and diagonal neighbours. All cells within the same antidiagonal can be computed in parallel forming a wavefront. A parallel Matrix Fill implementation is based on this wavefront approach to fill the matrices. Banded Matrix Fill is implemented as a typical systolic array architecture of Processing Elements (PEs) equal in number to the length of the read sequence. Each PE is responsible for computing a single row of the matrix in Fig.6.2. All PEs operate in parallel and compute all cells within the same antidiagonal per wavefront. As a result, the computed cells should be stored per antidiagonal rather than per row. Fig.6.6 depicts the PE array computations and the matrix required to store all antidiagonals. Each row in this matrix corresponds to a row of the original matrix and each column to an antidiagonal. Each one includes as many elements as the length of the longest antidiagonal, i.e. the read length.

Storing Optimization Scheme: Fig.6.6 also illustrates how the *band* elements are stored in a smaller matrix. PEs temporarily store each anti-diagonal in a read-length vector. Each antidiagonal includes at most $bandLength/2 + 1$ cells that belong to the banded area. Therefore we need to allocate a matrix of vectors, i.e.$bandVectors$, of length $bandLength/2 + 1$ rather than $bandLength$ to store all band cells. A mask of length $bandLength/2 + 1$ shifts through the read-length vector written by the PEs, selects the band cells and stores them in memory. Note that storing the bandRow vectors in a BRAM changes the relative position of some neighbor cells by an offset of 1.

Traceback Banded Design: Traceback receives these $bandVectors$ in reverse order and constructs the alignment path by keeping track of the first and last matching characters of the sequences and all potential edits in between. Fig.6.7 illustrates an abstract diagram of the logic implemented by Traceback. Traceback encodes each backward step in the alignment as one of the available distinct states: i) initial state ii) H-up, iii) H-left, iv)

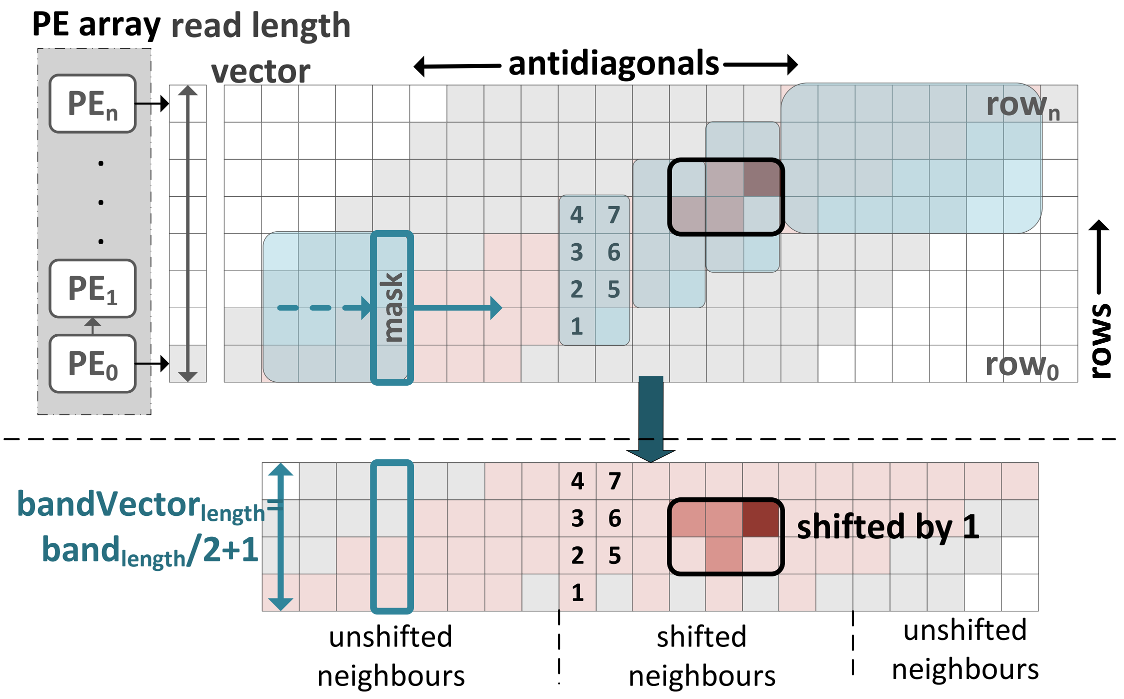


Figure 6.6.: PE computation of score matrices and BRAM allocation of band elements.

F-up, v) E-left, vi) H-diagonal, vi) *waiting* state and vii) *finished* state. In order to enable Traceback to decide its next state, i.e. the next hop of the backward path, we check all possible origins from neighboring cells in the two subsequent antidiagonals. These checks are taking place in parallel and result in setting a single bit of an 8-bit vector, i.e. one bit allocated for each possible state. This 8-bit vector then is leveraged as a selector signal in a set of parallel one-hot-multiplexers. Each one-hot-multiplexer links to each one of the internal Traceback variables, which actually define the Traceback's state space. Thus, the selector signal is utilized to update all identifiers of the next state such as the type of cell (E,F or H), the index within the read/reference sequence and the index within the bandRow vector.

Special attention is required with indexing when accessing the neighboring cells within the two subsequent vectors, as bandRow vectors have been shifted to be stored in BRAM. Since the two subsequent vectors are shifted by one position at most, we add a correction offset of value 1 when necessary.

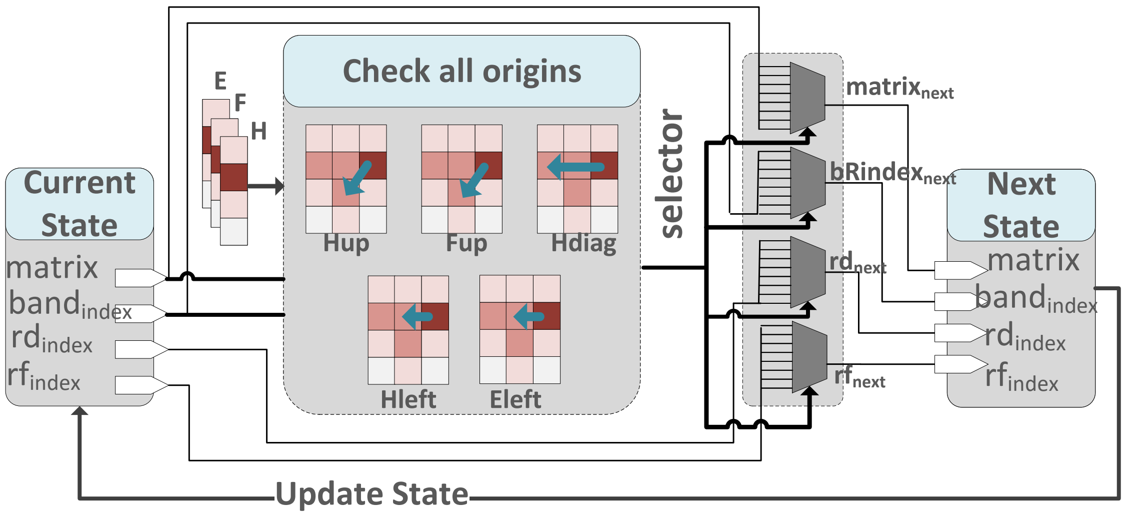


Figure 6.7.: Logic Diagram of Traceback Dataflow implementation.

6.4. Experimental Results

6.4.1. Experimental Setup

The experimental setup includes a dual socket Intel Xeon Gold 6138 (2.0GHz, 14nm) with 128GB DDR4 DRAM (2133MHz) that is connected through PCIe 2.0 with Maxeler's MAX5C platform. The specific MAX5C provides two MAX5 dataflow engines, i.e. two Xilinx VU9P Ultrascale FPGAs. Accelerators have been developed using MaxJ dataflow language [239] and compiled with MaxCompiler 2018.3 that invokes Vivado 2017.4 for place and route purposes.

6.4.2. Banded Smith-Waterman Evaluation

Accuracy Evaluation

The adoption of Banded SmithWaterman reduces the accuracy of the alignment by default, as it eradicates alignments that do not meet the edit threshold. This section evaluates the accuracy of short read alignment when applying pre-filtering and leveraging the proposed Banded SmithWaterman implementation for the respective edit threshold. As a reference, we utilize the results of Bowtie2 aligner. Bowtie2 aligner by default considers 150 as a maximum limit for number of edits, which is equivalent to a no-limit alignment strategy for the 100-base length of the given datasets. Pre-filtering is implemented by SneakySnake, that discards all candidates with more edits than the specified

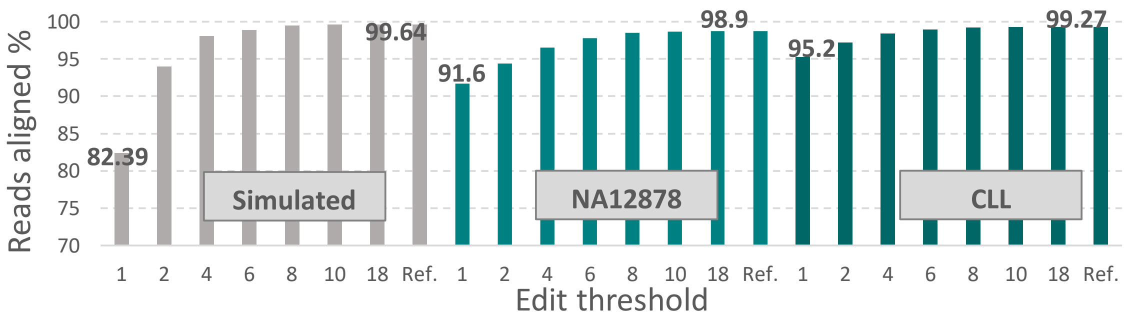


Figure 6.8.: Alignment success when using SneakySnake pre-filtering and the proposed Banded Smith-Waterman accelerator for seed extension.

edit threshold. We consider edit thresholds that vary from 1 up to almost 20% of the read length of 100. We evaluate the accuracy on the three datasets described in Section 6.2. The comparison is illustrated in Fig.6.8. The results are in accordance with the edit distribution findings, as for most datasets the alignment rate converges to the optimal one for an edit threshold greater than 10, which includes most of the alignments.

Performance Evaluation

The performance of the proposed accelerator is compared against both software and hardware state-of-the-art alignment implementations. The comparison is based on the throughput of all implementations, i.e. the number of computed alignments per seconds. We first consider a single-instance comparison strategy, which employs a single thread for the software aligners and a single acceleration instance for the proposed aligner. We utilize the following state-of-the-art software aligners: (i) Edlib [114], a C/C++ aligner that implements exact sequence alignment using the Levehnstein edit distance for seed extension, (ii) WFA [115], a software aligner that performs exact alignment based on a gap-affine scoring scheme and optimized with the wavefront parallelism approach, (iii) KSW2 [116], a C library that aligns pairs of biological sequences based on dynamic programming, (iv) the vectorized SSE2-based SmithWaterman implementation utilized in Bowtie2 that adopts a heuristic to support vectorization with 8 SIMD lanes [113], (v) the open-source software implementation of GenASM [136], an approximate string matching acceleration framework for genome sequence analysis. The single-instance proposed Banded SmithWaterman accelerator includes an 100-PE array and is built with a clock frequency of 400MHz. Since edit threshold only affects the resource utilization of the single-instance design, all edit threshold designs can be utilized to measure the performance. We utilize the Simulated dataset of 100-base long reads for execution. The results are found in Fig.6.9. The proposed Banded design acquires a speedup of $\times 34$ over the fastest software implementation, which is Edlib.

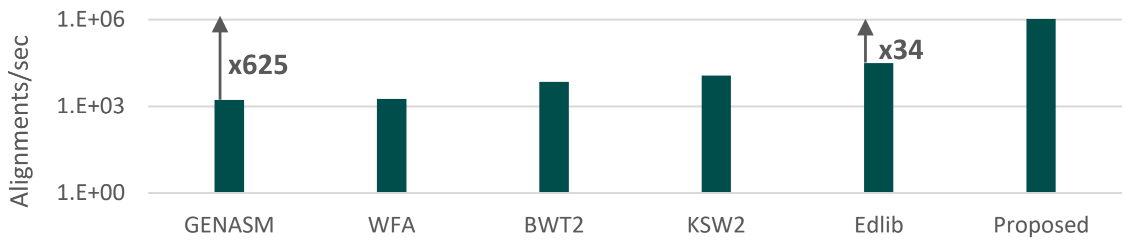


Figure 6.9.: Throughput comparison between Proposed Banded 10-edit threshold accelerator and state-of-the-art SW aligners.

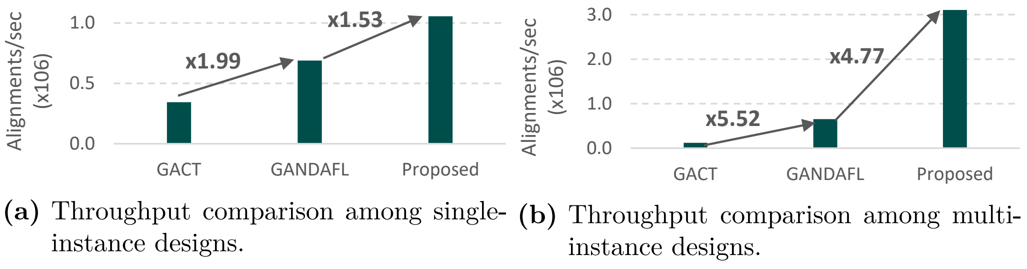


Figure 6.10.: Throughput and Resource utilization efficiency evaluation for HW Smith-Waterman accelerators.

We also compare our design with two state-of-the-art hardware accelerators, GACT [36] and GANDAFL [246]. GACT accelerator is part of Darwin short read aligner and implements an RTL PE-based design of SmithWaterman MatrixFill and Traceback. We utilize the open-source implementation of GACT and implement the hardware using Vivado HLS 2018.3. We developed an in-house implementation of GANDAFL, which is a dataflow MatrixFill and Traceback FPGA implementation that also targets Maxeler MAX5C workstation. We utilize MaxCompiler 2018.3 for development and synthesis purposes. We examine two different scenarios for a rounded comparison. The first one compares the throughput of a single instance of each accelerator, to evaluate the capability of the design to deliver a single alignment result faster than state-of-the-art solutions. The second one compares multi-accelerator designs that leverage the maximum capacity of the same target FPGA device. We utilize the FPGA included within MAX5C platform, i.e. Xilinx VU9P and assume that all designs share the same PCIe connection. Table 6.1 summarizes the optimal configurations (that minimize the Area $\times$ Delay product metric) selected for each accelerator for the examined scenarios. As seen in Fig.6.10, the Proposed Banded implementation achieves a $\times 1.53$ speedup over GANDAFL dataflow and a $\times 3$ speedup over GACT single instance designs. Thanks to efficient resource utilization, the proposed multi-dataflow design fits 8 accelerators on the FPGA and achieves a $\times 4.77$ speedup over the 3-dataflow instances of GANDAFL accelerator. The dataflow computing model and efficient resource utilization secure a $\times 26.35$ speedup over a 12-instance design of GACT.

Table 6.1.: Hardware Accelerators Configurations for throughput comparison.

Configuration			Utilization			Clock
Accelerator	Instances	PEs	BRAM18	DSP	Logic	(MHz)
GACT	1	128	3.10%	0.0%	2.39%	250
	25	128	77.55%	0.0%	60.71%	150
GANDAFL	1	100	24.75%	0.10%	8.68%	200
	3	100	65.53%	0.31%	22.62%	200
Proposed	1	100	11.78%	0.10%	7.60%	400
18 edits	8	100	53.94%	1.4%	55.44%	250

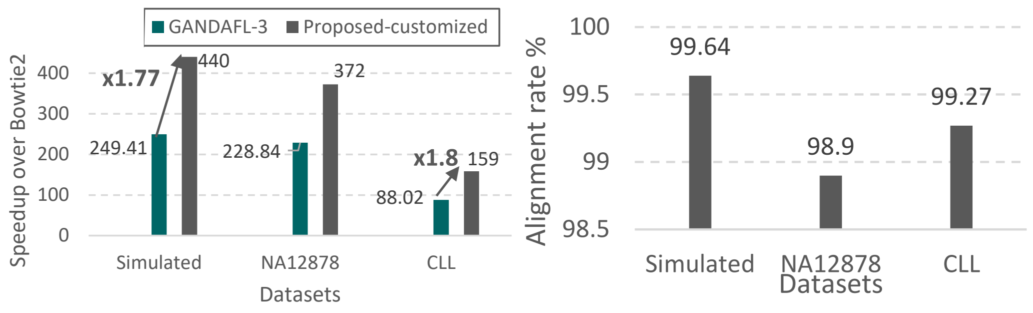


Figure 6.11.: Performance and accuracy evaluation of proposed customized accelerated system.

6.4.3. Multi-Dataflow System Evaluation

Based on the proposed methodology, we evaluate the performance of customized accelerated systems on the Simulated, NA12878 and CLL datasets. First we leverage the edit profiles and build the respective multi-dataflow designs for a clock of 250MHz. Table 6.2 summarizes the configurations and the percentage of alignments covered within each threshold selected. We compare these designs with our GANDAFL in-house implementation that includes 3 accelerator instances that allow up to 17 edits and is clocked at 200MHz. Both designs are also compared with Bowtie2 aligner, which generates the input alignments and serves as a reference for the alignment rate. SneakySnake pre-filtering has been leveraged to classify the candidate alignments and assign them to the respective dataflow accelerator based on the edit threshold. As seen in Fig.6.11, the proposed approach delivers a speedup up to ×440 over Bowtie2 aligner and ×1.8 over GANDAFL conventional accelerated approach. Note that this speedup does not come at the expense of alignment rate as the alignment rate has converged to the reference one for the selected thresholds.

Table 6.2.: Multi-Dataflow Configurations customized to the edit profiles of input datasets.

Dataset	Configuration		
	Edit threshold	Instances	Alignments covered
Simulated	1-2-10-18	5-1-1-1	70.8-81.4-99.7-99.9%
NA127828	1-2-5-18	5-1-1-1	74.4-87.3-99.6-99.9%
CLL	1-5-18	6-1-1	82.5-93.51-99.9%

Chapter 7.

Conclusions

This chapter briefly summarizes the novelties and results of our proposed implementations and discusses the conclusions derived from the Ph.D. Thesis. We then present potential improvements and future extensions based on the current trends and requirements of the fields.

7.1. Summary of Ph.D. Thesis

Technological advancements and novelties in the last decade have led to an exponential growth of data and signaled the start of the Big Data Era. This onslaught of information and data and the need to extract value from raw data in order to provide new services has created great business opportunity and has led to the rising of the big data market. Healthcare holds a considerable portion of the market share as big data analytics are required to cope with an exponential data growth attributed to the digitization of healthcare data and the advent of new technologies. A major source of healthcare data area generated during clinical practice (e.g. ECG,EEG, X-ray, CT scans, MRI, SPECT) as well as from IoT devices, e.g. health-tracking wearable devices e.t.c. A huge part of health data also come from the advent of whole-genome sequencing and other high-throughput molecular technologies that have created the data rich disciplines of genomics, transcriptomics, epigenomics, proteomics, metabolomics, phenomics e.t.c., i.e. *omics* data.

A wide range of healthcare big data analytics are developed to extract value out of this immense and diverse amount of data. When handling signals and images, advanced signal and image processing techniques are required that stem from the fields of computer vision and pattern recognition. Machine learning techniques and AI are also efficiently incorporated in workflows to perform detection and prediction tasks, thanks to their ability to discover patterns and correlations within complex data. This has led to a broad development of predictive analytics, that guide the medical staff in making a diagnosis and choosing effective treatments based on previous experience and results. Complex computational and statistical methodologies are also leveraged to manage omics data and understand complex mechanisms on a cellular or even metabolic level. Genome analysis is at the heart of omics workflows as it is an essential tool to understand how DNA variants are expressed and alter phenotypes. Read alignment is performed in the beginning of genomic pipelines to reconstruct the genome of a sample and compare it to the reference genome for variant discovery using some kind of string matching algorithm (e.g. SmithWaterman). The output of genome analysis can be combined with other *omics* studies, generating knowledge that can be used for research and clinical purposes, paving the way to personalized medicine.

The size of this data however is continuously growing, as is the complexity of the algorithms developed to handle them. This stress the limits of current systems and highlights the need for optimization. An initial approach for addressing this challenge was the development of big data infrastructures and frameworks for distributed computation (e.g. HDFS, MapReduce). More recently, the execution of healthcare analytics have also mitigated to the cloud which provides virtualized services, scalability and reliability at low cost. However, it is not always possible to upload data to the cloud, as often ethics and privacy matters occur especially in the healthcare domain. Hardware acceleration is an effective alternative to cope with the data and compute intensive applications of the field. GPUs, FPGAs and powerful co-processors in general have been employed for building accelerated systems for both machine learning prediction models and DNA aligners in genomic workflows.

In this thesis, we present our efforts at creating efficient accelerators for a Support Vector Machine classifier for ECG arrhythmia detection and Short Read Alignment on Next Generation Sequencing data, leveraging *High Level Synthesis Techniques* and targeting *FPGA* devices.

An Exploration Framework for Efficient High-Level Synthesis of Support Vector Machines: In this work we present a methodology for creating efficient HLS based HW accelerators targeting Support Vector Machine based classifiers. The proposed methodology relies on two levels. The first level optimizes the original code under acceleration in order to assist the HLS tool to maximize the parallelization of the computational parts of the algorithm and infer data- as well as instruction- level parallelism. The second level relies on the built-in HLS directives and evaluates combinations that lead to efficient architectures. In order to avoid an exhaustive search of the search space, we propose pruning guidelines that are based on the algorithmic structure and memory access patterns of the SVM and generate a space whose pareto front is close to the original one.

GANDAFL: Dataflow Acceleration for Short Read Alignment on NGS data: This work focuses on accelerating the short read alignment of Next Generation sequencing data which is a major bottleneck in genomic pipelines. Accelerated reconfigurable computing has been extensively leveraged to alleviate this bottleneck, focusing mostly on high-performance implementations that do not take into account the implications of integrating the accelerated part of an aligner within the sequencing tool. As a result, aspects such as system wide communication and accelerator call overheads are neglected. In this work, we address the aforementioned inefficiencies and propose GANDAFL, a novel genome alignment dataflow architecture for SmW Matrix-fill and Traceback stages to perform high throughput short-read alignment on NGS data. We then propose a radical software restructuring to widely-used Bowtie2 aligner that allows read alignment by batches to expose acceleration capabilities. Batch alignment minimizes calling overhead of the accelerators whereas moving both Matrix-fill and Traceback on chip extinguishes the communication data overheads. The standalone solution delivers up to $\times 116$ and $\times 2$ speedup over state-of-the-art software and hardware accelerators respectively and GANDAFL-enhanced Bowtie2 aligner delivers a $\times 1.9$ speedup.

Profile-Driven Banded Smith-Waterman acceleration for Short Read Alignment: In this work, we employ hardware acceleration and pre-filtering methods to present a profile-driven Smith-Waterman accelerated design for short read alignment. Extensive profiling of genomic datasets reveals low edit thresholds that can be leveraged by Banded SmithWaterman to create resource-efficient accelerators that are customized to the edit profile of the input. We therefore design and deliver a highly optimized dataflow implementation for Banded Smith-Waterman seed-extension targeting FPGA devices, which is leveraged within a multi-dataflow accelerated system. This system is configured with multiple Banded accelerators covering the full range of edits to achieve both high throughput as well as high accuracy alignment. The evaluation shows that the proposed Banded Smith-Waterman accelerator delivers a $\times 34$ speedup over state-of-the-art software aligner and $\times 1.53$ over state-of-the-art dataflow SmithWaterman accelerator [246] and $\times 3$ over GACT RTL accelerator [36]. Moreover, the proposed multi-dataflow system delivers average speedups of $\times 1.8$ over state-of-art multi-accelerator FPGA solutions [246] that employ generic and input-agnostic accelerators.

7.2. Future Extensions

The ever-growing healthcare data size and the increasing complexity of ML-healthcare and Genomic applications have led to many open problems and call for constant contributions and optimizations. Within the context of this thesis, we could pursue several directions that are expected to bring promising results:

- **Further optimization of GANDAFL:** GANDAFL is a highly optimized accurate version of the SmithWaterman string matching algorithm. The integration with Bowtie2 however is constrained in terms of end to end speedup by the Amdhal law. An alternative course would be to extend the part of the aligner implemented on hardware and also implement in dataflow the seeding step of Bowtie2 aligner.

This would avoid the integration implications and lead to a hardware-only aligner design that is no longer bounded by Amdahl-law for speedup gains. An important challenge in this approach is to maintain the compatibility with the CIGAR format and reported output of Bowtie2 aligner.

- **Optimization of Genomic Pipeline in Cloud environment:** A genomic pipeline comprises of many data and compute intensive steps. For example, after read alignment, typically follows the variant calling stage. In fact, multiple variant callers are often utilized within a single workflow. As the cloud environment has proved an efficient solution for big data applications, there is increased interest in extensively exploring how the resources of a cloud environment could improve the performance of an entire genomic pipeline. The exploration could also leverage the power of GPUs and FPGAs (i.e. the existing accelerator and any open-source available ones) and generate guidelines and insights for efficient execution.

- **Serverless Genomics:** In the context of optimizing an end-to-end genomic pipeline, it would be interesting to achieve this goal using the serverless computing model. In this new computing and scheduling model, large applications are transformed into more structured ones with smaller execution units. Each of these smaller tasks is scheduled to servers and assigned resources depending on the availability and requirements. Deployment of tasks is decided by the framework upon a user request for a workflow execution. Once deployed, the tasks communicate with each other in an event-driven manner, without needless interactions with the framework. A genomic pipeline could be broken into multiple such smaller tasks and execute on a cloud platform, leveraging all available computing resources including FPGA or GPU accelerators.

- **Leveraging Machine Learning in Genomics Workflows:** This thesis individually examines a healthcare application that relies on a machine learning model and a genomic application, i.e. short read alignment. There is however great potential in a synergetic approach that leverages Machine learning within the genomics domain. This is supported by an elaborate review of the use of deep learning in genomics [251]. The authors in [251] argue that the use of deep learning in genomics has multiple advantages: (i) it can replace multiple time-consuming pre-processing steps with a single model thanks to its inherent ability to discover patterns within complex data, (ii) it can handle heterogeneous data of different origin and type effectively (e.g. RNA and image data), (iii) it is ideal for the detection of spatial patterns and (iv) it provides an abstraction layer over complex statistic and mathematical formulation, and therefore it can increase productivity. There is already a wide use of deep learning in various genomics applications, coming from supervised, multimodal, transfer as well as unsupervised learning domains and applied to tasks such as gene expression profile prediction [252], prediction of noncoding variants [253], base-calling [254] e.t.c. As the application of deep learning continues to expand across multiple omics data types and gradually becomes part of everyday clinical practice, the research interest in new models is expected to spike. This increase in applications and the continuous growth of omics data will be most definitely accompanied with the adoption of numerous High Performance optimization techniques. The authors in [255] present an extensive list of deep learning optimization frameworks that will be most likely leveraged in the genomics fields. Recent works also align with this trend, e.g. authors in [256] present a deep convolutional neural network toolkit for epigenomics that have been trained on GPUs whereas the work in [257] delivers a hardware-optimized deep-learning-based genomic basecaller.

Chapter 8.

Συνοπτική Περιγραφή των Προτεινόμενων Μεθοδολογιών στα Ελληνικά

Σε αυτή τη διατριβή επικεντρωνόμαστε στην υλοποίηση υλικού επιτάχυνσης για δύο αντιπροσωπευτικές εφαρμογές του σύγχρονου τομέα της υγείας: μια ανάλυση πρόβλεψης που βασίζεται στη μηχανική μάθηση και η ευθυγράμμιση ανάγνωσης γονιδιωματικών δεδομένων. Και οι δύο τομείς βιώνουν έντονη ανάπτυξη τις τελευταίες δεκαετίες και παράγουν έναν τεράστιο όγκο ακατέργαστων δεδομένων, πλούσιο σε πληροφορία. Η ερμηνεία και η λήψη αποφάσεων βασισμένων σε αυτά τα δεδομένα έχουν αποδειχθεί δύσκολες εργασίες καθώς τα δεδομένα και η υπολογιστική πολυπλοκότητα των αλγορίθμων αυξάνονται εκθετικά. Για να αντιμετωπιστεί αυτό το πρόβλημα, έχουν εξεταστεί τεχνικές υψηλής απόδοσης όπως η επιτάχυνση σε ηαρδωαρε. Υπάρχει μια πληθώρα ερευνητικών εργασιών που αξιοποιούν διαφορετικά μοντέλα προγραμματισμού για να αναπτύξουν αποτελεσματικούς επιταχυντές βασισμένους σε FPGA, χάρη στην ευελιξία προγραμματισμού τους σε επίπεδο βιτ. Ωστόσο, τα διαθέσιμα μοντέλα προγραμματισμού για την προγραμματισμό τέτοιων συσκευών δεν μπορούν πάντα να εκμεταλλευτούν πλήρως τις προοπτικές επιτάχυνσης των εφαρμογών με απλό τρόπο. Επιπλέον, σε πολύπλοκες εφαρμογές, οι υπάρχουσες λύσεις χαρακτηρίζονται από μια περιορισμένη οπτική στην ενσωμάτωση των επιταχυντών σε ένα ρεαλιστικό σύστημα, όπως η επικοινωνία σε επίπεδο συστήματος και οι πρόσθετοι χρόνοι κλήσης των επιταχυντών. Στο τρέχον διδακτορικό, η κύρια συνεισφορά βασίζεται στην παροχή αποτελεσματικών λύσεων μέσω της στρατηγικής εξερεύνησης του χώρου σχεδιασμού και της συνέργιας βελτιστοποιήσεων του κώδικα τόσο σε επίπεδο υλικού όσο και λογισμικού.

Η πρώτη εφαρμογή που εξετάζεται σε αυτή τη διατριβή είναι η αποδοτική επιτάχυνση υλικού των ταξινομητών Συππορτ ἕςτορ Μαςηινε (SVM). Σε αυτήν τη διατριβή, εξετάζουμε μια εφαρμογή στην οποία οι επιταχυντές υλικού SVM εκτελούν ταξινόμηση για την ανίχνευση αρρυθμιών σήματος ECG. Η προτεινόμενη μεθοδολογία για την επιτάχυνση του SVM έχει υλοποιηθεί χρησιμοποιώντας το εργαλείο Vivado High-Level Synthesis (HLS). Προτείνουμε μια συστηματική προσέγγιση δύο επιπέδων για την επιτάχυνση του SVM, η οποία πρώτα βελτιστοποιεί τη γενική δομή της αρχικής περιγραφής συμπεριφοράς του SVM για να βοηθήσει το εργαλείο να αναγνωρίσει τον εγγενμή παραλληλισμό σε επίπεδο δεδομένων και εντολών του αλγορίθμου. Το δεύτερο επίπεδο βελτιστοποίησης βελτιώνει επιπρόσθετα το σχεδιασμό μέσω μιας στρατηγικής εξερεύνησης του χώρου σχεδιασμού που σχεδιάζει τη μνήμη του επιταχυντή βάσει των μοτίβων υπολογισμού και πρόσβασης στη μνήμη του.

Στο δεύτερο μέρος της διπλωματικής εργασίας, μελετάμε την επίδραση των τεχνικών επιτάχυνσης σε ένα από τα πιο υπολογιστικά απαιτητικά κομμάτια της επεξεργασίας γονιδιώματος, που είναι η ευθυγράμμιση ακολουθιών ΔΝΑ στο ανθρώπινο γονιδίωμα. Εκτελούμε ανάλυση της απόδοσης ενός εργαλείου αλληλούχισης (το Bowtie2) και εντοπίζουμε τον αλγόριθμο Smith-Waterman ως το πιο χρονοβόρο κομμάτι. Η προσέγγισή μας είναι να παρέχουμε μια υλοποίηση ροής δεδομένων που στοχεύει συσκευές FPGA λαμβάνοντας υπόψη τις συνέπειες της ενσωμάτωσης του επιταχυντή στο εργαλείο αλληλούχισης και επομένως σε ένα πραγματικό σύστημα. Προτείνουμε το GANDAFL, μια νέα αρχιτεκτονική ροής δεδομένων ευθυγράμμισης γονιδιώματος για τον Smith-Waterman για την εκτέλεση ευθυγράμμισης υψηλής απόδοσης σε δεδομένα αλληλουχίας επόμενης γενιάς. Στη συνέχεια, προτείνουμε μια ριζική αναδιάρθρωση του κώδικα του Bowtie2 η οποία ομαδοποιεί πολλά μεμονωμένα αιτήματα αλληλούχισης και τα τροφοδοτεί στον επιταχυντή με υψηλής ρυθμό απόδοσης ελαχιστοποιώντας έξοδα μεταφοράς και κλήσεων. Ο επιταχυντής προσφέρει έως και 116 και 2 φορές επιτάχυνση αντίστοιχα σε σύγκριση με πρόσφατους επιταχυντές λογισμικού και υλικού, αντίστοιχα, και η βελτιωμένη με GANDAFL ευθυγράμμιση Bowtie2 προσφέρει επιτάχυνση 1,9 επί του συνολικού συστήματος. Τέλος εξετάζουμε μια εναλλακτική προσέγγιση, η οποία συνδυάζει μια ευριστική υλοποίηση του Smith-Waterman και ένα στάδιο φιλτραρίσματος των αρχικών δεδομένων. Μελέτη των δεδομένων εισόδου υποδεικνύει ότι η αλληλούχιση συνήθως είναι ακριβής και εντοπίζεται μικρός αριθμός διαφοροποιήσεων από το ανθρώπινο γονιδίωμα. Αυτό μειώνει το χώρο αναζήτησης των λύσεων και μας επιτρέπει να χρησιμοποιήσουμε τον ευριστικό Banded Smith Waterman ο οποίος επιτελεί την ίδια λειτουργία, εντοπίζει λιγότερες διαφοροποιήσεις και καταναλώνει λιγότερους πόρους στο υλικό. Προτείνουμε λοιπόν ένα σύστημα που πλέον αποτελείται από πολλούς επιταχυντές και καλύπτει έως έναν αριθμό διαφοροποιήσεων ενώ εντοπίζει πλέον τις αλληλουχίσεις με ταχύτερο ρυθμό. Το προτεινόμενο σύστημα αποδίδει επιτάχυνση έως 34 φορές σε σχέση με λογισμικά ενώ είναι έως 3 φορές γρηγορότερο από σχετικούς επιταχυντές.

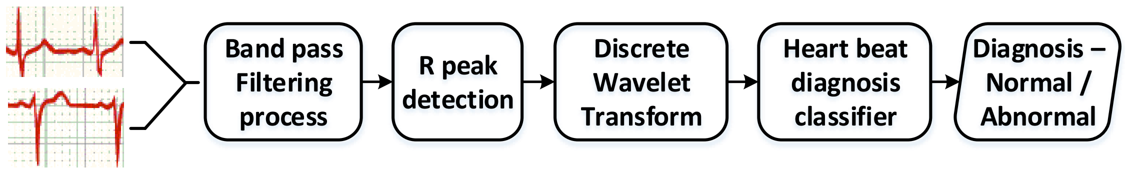
8.1. Μέθοδος διερεύνησης χώρου λύσεων για Αποδοτική Υψηλού Επιπέδου Σύνθεση Support Vector Machine

Η ενότητα αυτή βασίζεται στη δημοσίευση μας [180].

Στόχος αυτού του κεφαλαίου είναι η αξιοποίηση των δυνατοτήτων του HLS για τη δημιουργία αποδοτικών SVM ως επιταχυντές σε υλικό. Μελετάται ο εντοπισμός αρρυθμιών στο ηλεκτροκαρδιογράφημα ΗΚΓ χρησιμοποιώντας ως βάση δεδομένων μια βάση δεδομένων για ΗΚΓ που έχει αναπτυχθεί μέσω κοινής συνεργασίας των πανεπιστημίων MIT και BIH. Σε πρώτο επίπεδο ο αρχικός κώδικας αναδομείται με κριτήριο την επιτάχυνση ώστε να δημιουργηθεί αποδοτικός επιταχυντής. Σε δεύτερο επίπεδο εξερευνώνται οι τεχνικές βελτιστοποίησης του εργαλείου HLS οι οποίες εφαρμόζονται στον αρχικό και στον τροποποιημένο κώδικα για περαιτέρω βελτίωση του ως προς μετρικές επίδοσης και χρησιμοποίησης πόρων. Προτείνεται στρατηγική αποδοτικής εξερεύνησης του χώρου λύσεων ώστε να δοθούν στο σχεδιαστή τα βέλτιστα σημεία κατά Pareto με βάση τα οποία μπορεί να επιλέξει μια υλοποίηση ανάλογα με τις απαιτήσεις της εκάστοτε εφαρμογής σε ταχύτητα εκτέλεσης και χρησιμοποίηση πόρων.

Εφαρμογή Εντοπισμού Αρρυθμίας σε Ηλεκτροκαρδιογράφημα με χρήση Support Vector Machine Η μορφή του ΗΚΓ και ο καρδιακός ρυθμός που εξάγεται από το ΗΚΓ είναι δηλωτικά της κατάστασης της καρδιάς. Στην ουσία το ΗΚΓ αποτυπώνει διαδοχικούς καρδιακούς κύκλους. Ο καρδιακός κύκλος (διαστολή, συστολή, ηρεμία) συντονίζεται από ηλεκτρικά σήματα που παράγονται από κατάλληλα κέντρα διέγερσης της καρδιάς. Υπάρχουν τρία βασικά ηλεκτρικά σήματα που εμφανίζονται στο ΗΚΓ: το έπαρμα P, το σύμπλεγμα QRS που αποτελείται από τις κορυφές Q,R,S και το έπαρμα T. Αυτά τα σήματα είναι στην πραγματικότητα μεταβολές του ηλεκτρικού δυναμικού διαφόρων περιοχών της καρδιάς και άρα το ΗΚΓ απεικονίζει την ηλεκτρική δραστηριότητα της καρδιάς. Τα επάρματα αυτά και οι αποστάσεις μεταξύ τους έχουν συγκεκριμένη χρονική διάρκεια και μορφολογία. Οποιαδήποτε παρέκκλιση από τη φυσιολογική μορφολογία τους πρέπει να μελετηθεί καθώς μπορεί να είναι δείγμα παθολογικής βλάβης. Η καρδιακή αρρυθμία είναι η πιο συνηθισμένη καρδιακή βλάβη και είναι η διαταραχή του καρδιακού ρυθμού. Η αρρυθμία μπορεί να είναι από ασυμπτωματική μέχρι κρίσιμη για την ανθρώπινη ζωή. Για αυτό το λόγο κρίνεται απαραίτητη η μελέτη του ΗΚΓ, ως μέσο διάγνωσης αρρυθμιών. Οι αρρυθμίες είναι μεμονωμένα περιστατικά που εκδηλώνονται σε τυχαίες χρονικές στιγμές. Επομένως είναι αναγκαία η μελέτη του ΗΚΓ μεγάλων χρονικών διαστημάτων. Ο μεγάλος όγκος δεδομένων προς μελέτη καθιστά απαραίτητη τη χρήση τεχνικών μηχανικής μάθησης για την επεξεργασία του. Ταξινομητές βασίζονται σε τεχνικές μηχανικής μάθησης για την εκπαίδευσή τους με αυτό το μεγάλο σύνολο δεδομένων ώστε τελικά να μπορούν να διαγνώσουν σωστά την ύπαρξη ή μη αρρυθμίας σε ένα νέο σύνολο δεδομένων ΗΚΓ. Στη συγκεκριμένη εργασία χρησιμοποιείται η βάση δεδομένων αρρυθμίας MIT-BIH Arrhythmia Database, η οποία περιλαμβάνει παλμούς για τους οποίους έχει γίνει διάγνωση από καρδιολόγους. Η διαδικασία επεξεργασίας και ανάλυσης του ΗΚΓ για την εξαγωγή των επιμέρους παλμών και των χαρακτηριστικών τους ώστε τελικά να γίνει η διάγνωση χρησιμοποιώντας μοντέλα τεχνικής μηχανικής μάθησης παρουσιάζεται ακολούθως και απεικονίζεται στο Σχ.8.1.

Ως ταξινομητής επιλέγονται οι Μηχανές Διανυσμάτων Υποστήριξης (Support Vector Machines -SVM). Τα SVM είναι μοντέλα επιβλεπόμενης μάθησης που εκπαιδεύονται με ένα μεγάλο σύνολο δεδομένων και είναι κατάλληλη για την ταξινόμηση των νέων εισόδων σε δύο υποψήφιες κλάσεις συμπληρωματικές μεταξύ τους. Το σύνολο εκπαίδευσης αποτελείται από διανύσματα με συγκεκριμένα χαρακτηριστικά καθένα από τα οποία διαθέται μια ετικέτα


Band pass Filtering process
R peak detection
Discrete Wavelet Transform
Heart beat diagnosis classifier
Diagnosis – Normal / Abnormal

Σχήμα 8.1.: Ροή Ανάλυσης Ηλεκτροκαρδιογραφήματος.

δηλωτικής της κλάσης στην οποία ανήκει. Ένα σύνολο από άλλα διανύσματα με τα ίδια χαρακτηριστικά και γνωστές τις ετικέτες χρησιμοποιείται για να ελεχθεί η ακρίβεια της πρόβλεψης.

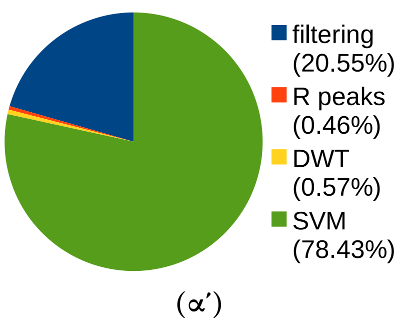
Τα SVM εφαρμόζουν αρχικά μια συνάρτηση πυρήνα που ανάγει τα διανύσματα σε ένα χώρο περισσότερων διαστάσεων, όπου είναι πιο εύκολος ο διαχωρισμός τους. Στο χώρο αυτό βρίσκουν ένα υπερεπίπεδο το οποίο αποτελείται από τα διανύσματα που απέχουν μέγιστα από τα διανύσματα που ανήκουν σε κάθε κλάση. Κάθε νέο διάνυσμα ανάγεται σε αυτόν το χώρο, υπολογίζεται η απόσταση του από το υπερεπίπεδο και άρα με βάση τη θέση του σε σχέση με αυτό ταξινομείται στην αντίστοιχη κλάση. Η συνάρτηση πυρήνα είναι καθοριστική για την ακρίβεια και την πολυπλοκότητα του μοντέλου. Λόγω των μη γραμμικών σχέσεων μεταξύ των χαρακτηριστικών του διανύσματος κάθε παλμού χρησιμοποιούμε μη γραμμική συνάρτηση πυρήνα και συγκεκριμένα εκθετικής φύσης.

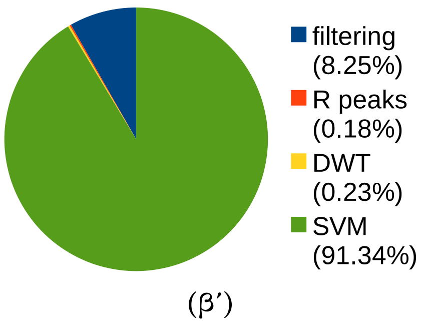
Ακολουθεί η μαθηματική εξίσωση που περιγράφει τον υπολογιστικό πυρήνα του ταξινομητή και ο αντίστοιχος κώδικας C που την υλοποιεί:

$$Class = sgn(\sum_{i=1}^{N\_sv} (y_i * a_i * exp(-\gamma ||\mathbf{x} - \mathbf{sup\_vector}_i||^2)) - b) \qquad (8.1)$$

Το σχήμα 8.2 απεικονίζει τη μέση χρήση της CPU ανά επεξεργασία καρδιακού παλμού για τα διαφορετικά στάδια επεξεργασίας της ροής ανάλυσης ΗΚΓ που εκτελείται σε ένα Intel Quark SoC. Διαφορετικά μοντέλα ταξινομητών SVM, με αυξανόμενες υπολογιστικές απαιτήσεις (όσον αφορά τον αριθμό των διανυσμάτων υποστήριξης και το μέγεθος του διανύσματος εισόδου) χρησιμοποιήθηκαν κατά το profiling της εφαρμογής. Σε όλες τις περιπτώσεις οι είσοδοι της συσκευής είναι σήματα που προέρχονται από τη βάση δεδομένων αρρυθμιών MIT-BIH. Το Σχ.8.2α΄ δείχνει ότι ακόμη και για μοντέλα SVM μέτριας πολυπλοκότητας, η εκτέλεση του ταξινομητή κυριαρχεί στον απαιτούμενο χρόνο CPU.

Ο στόχος αυτής της δουλειάς είναι η δημιουργία ενός SW/HW συστήματος που εκτελεί την εφραμογή εντοπισμού αρρυθμιών πιο αποδοτικά υλοποιώντας την ταξινόμηση με το SVM σε έναν επιταχυντή σε FPGA. Τα χαρακτηριστικά του SVM που θα χρησιμοποιηθεί για





Σχήμα 8.2.: Μέση χρήση της CPU ανά επεξεργασία καρδιακού παλμού για (α) SVM μοντέλα μέτριων υπολογιστικών απαιτήσεων, (β) SVM μοντέλα υψηλών υπολογιστικών απαιτήσεων.

Πίνακας 8.1.: Παράμετροι του μοντέλου.

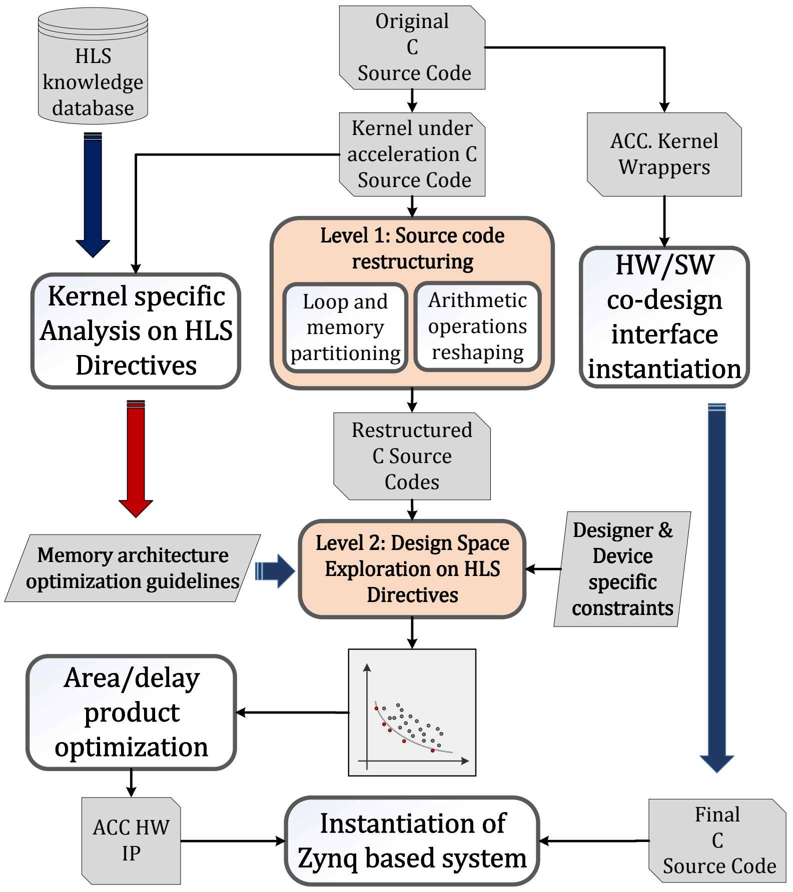
parameter	meaning	value
N_sv	number of support vectors	1274
D_sv	dimension/features of each support vector	18

την παρουσίαση της προτεινόμενης μεθοδολίας υλοποίησης αποδοτικών SVM σε FPGA περιλαμβάνονται στον Πίνακα 8.1.

Το Σχ.8.3 απεικονίζει την προτεινόμενη μεθοδολογία που στηρίζεται σε δύο επίπεδα βελτιστοποίησης. Στο πρώτο επίπεδο ο αρχικός κώδικας αναδομείται με κριτήριο την επιτάχυνση ώστε να διευκολύνει το εργαλείο HLS να εντοπίσει και να αξιοποιήσει τον υπάρχοντα παραλληλισμό. Στο δεύτερο επίπεδο εξερευνώνται οι τεχνικές βελτιστοποίησης του εργαλείου HLS οι οποίες εφαρμόζονται στον αρχικό και στον τροποποιημένο κώδικα για περαιτέρω βελτίωση του ως προς μετρικές επίδοσης και χρησιμοποίησης πόρων. Συγκεκριμένα, προτείνεται στρατηγική αποδοτικής εξερεύνησης του χώρου λύσεων ώστε να δοθούν στο σχεδιαστή τα βέλτιστα σημεία κατά Pareto με βάση τα οποία μπορεί να επιλέξει μια υλοποίηση ανάλογα με τις απαιτήσεις της εκάστοτε εφαρμογής σε ταχύτητα εκτέλεσης και χρησιμοποίηση πόρων.

Επίπεδο Βελτιστοποίησης 1: Αναδόμηση του HLS κώδικα.

Ανάπτυξη Παραλληλισμού σε Επίπεδο Μπλοκ: Αρχικά επιδιώκουμε την εξαγωγή παραλληλισμού σε επίπεδο συνάρτησης. Για να το επιτύχουμε αυτό εκμεταλλευόμαστε τον εγγενή παραλληλισμό του αλγορίθμου. Το διάνυσμα εισόδου του τρέχοντος παλμού υλοποιείται ως ένας πίνακας-γραμμή με 18 στοιχεία-χαρακτηριστικά. Τα διανύσματα υποστήριξης υλοποιούνται ως ένας δισδιάστατος πίνακας με τόσες στήλες όσο το πλήθος των διανυσμάτων υποστήριξης ενώ κάθε στήλη έχει τόσα στοιχεία όσα είναι τα χαρακτηριστικά που μελετώνται. Για κάθε διάνυσμα εισόδου προς ταξινόμηση, υπολογίζεται η ευκλείδια απόστασή του από κάθε διάνυσμα υποστήριξης και υψώνεται στο τετράγωνο. Στη συ-


HLS knowledge database
Original C Source Code
Kernel under acceleration C Source Code
ACC. Kernel Wrappers
Level 1: Source code restructuring
Loop and memory partitioning
Arithmetic operations reshaping
Kernel specific Analysis on HLS Directives
HW/SW co-design interface instantiation
Restructured C Source Codes
Memory architecture optimization guidelines
Level 2: Design Space Exploration on HLS Directives
Designer & Device specific constraints
Area/delay product optimization
ACC HW IP
Instantiation of Zynq based system
Final C Source Code

Σχήμα 8.3.: Προτεινόμενη ροή σχεδίασης HLS επιταχυντών για HW.

νέχεια εφαρμόζεται σε αυτή τη τιμή η συνάρτηση πυρήνα και η νέα τιμή που προκύπτει πολλαπλασιάζεται με τον αντίστοιχο παράγοντα κάθε διανύσματος υποστήριξης. Οι τιμές που προκύπτουν από τους υπολογισμούς με κάθε διάνυσμα υποστήριξης αθροίζονται και το τελικό αποτέλεσμα συγκρίνεται με τη τιμή βιας για την ταξινόμηση σε μια από τις δύο κλάσεις. Οι πράξεις που απαιτούνται μεταξύ του διανύσματος εισόδου και κάθε διανύσματος υποστήριξης είναι ανεξάρτητες μεταξύ τους. Μπορούν λοιπόν να εκτελούνται παράλληλα. Σε αυτή την έμφυτη παραλληλία βασίζεται η προτεινόμενη τεχνική. Ο πίνακας των διανυσμάτων υποστήριξης μπορεί να επιμεριστεί σε μικρότερους πίνακες, καθένας από τους οποίους περιέχει λιγότερα διανύσματα υποστήριξης. Οι πράξεις για τον υπολογισμό του μερικού αθροίσματος με το οποίο συνεισφέρει το κάθε κομμάτι πίνακα στο τελικό άθροισμα εκτελούνται παράλληλα. Επιτύχαμε λοιπόν την εκτέλεση του ίδιου υπολογιστικού πυρήνα πολλές φορές παράλληλα μόνο που κάθε μία από αυτές δρα σε μικρότερο σύνολο δεδομένων.

Η υλοποίηση της παραπάνω ιδέας απαιτεί αλλαγές στον κώδικα σε δομικό επίπεδο αλλά και τη χρήση των τεχνικών βελτιστοποίησης που προσφέρει το HLS. Συγκεκριμένα ο υπολογιστικός πυρήνας του ταξινομητή υλοποιείται ως συνάρτηση η οποία καλείται από την κύρια συνάρτηση τόσες φορές όσες φορές έχει επιμεριστεί ο πίνακας. Ο πίνακας των διανυσμάτων υποστήριξης και ο πίνακας των παραγόντων τους επιμερίζονται επίσης σε υποπίνακες με χρήση των κατάλληλων αυτόματων τεχνικών που παρέχει το εργαλείο. Σε διαφορετική περίπτωση θα δημιουργούνταν αντίτυπα των πινάκων για να είναι εφικτή η πρόσβαση σε πάνω από δύο στοιχεία του κάθε πίνακα τη φορά, περιορισμός που επιβάλλεται λόγω της υλοποίησης των πινάκων ως BRAM με δύο θύρες ανάγνωσης. Κάθε στιγμιότυπο της συνάρτησης έχει πρόσβαση στα στοιχεία μόνο ενός μέρους του επιμερισμένου πίνακα.

Αυτή η ιδέα υλοποιήθηκε για επιμερισμό του πίνακα σε 2,3,4,8 και 16 μέρη. Η βελτίωση του latency ήταν η αναμενόμενη, δηλαδή ο χρόνος εκτέλεσης διαιρέθηκε σχεδόν κατά έναν παράγοντα 2,3,4,8 και 16. Η χρησιμοποίηση σε DSP πολλαπλασιάστηκε κατά αυτόν τον παράγοντα ενώ υπήρχε σταδιακή αύξηση και στη χρησιμοποίηση LUT και Flip Flop. Η μνήμη παρέμεινε σταθερή εκτός από την τελευταία περίπτωση όπου σημειώθηκε μια απότομη αύξηση. Δοκιμάζοντας να χωρίσουμε τους πίνακες με το χέρι, δηλώνοντας τους εξαρχής χωριστά, πετύχαμε ακόμα μεγαλύτερη επιτάχυνση (ακόμα πιο κοντά στον ιδανικό παράγοντα 2,3,4,8,16 αντιστοίχως) και εξαλείφθηκε το πρόβλημα με την απότομη αύξηση σε BRAM.

Ανάπτυξη Παραλληλισμού σε Επίπεδο Εντολών μέσω μετασχηματισμού αριθμητικών υπολογισμών

Σε αυτό το κομμάτι θα εξετάσουμε την παραλληλοποίηση σε επίπεδο εντολών. Συγκεκριμένα θα ασχοληθούμε με την παραλληλοποίηση του εσωτερικού βρόχου του ταξινομητή. Αυτός ο βρόχος είναι υπεύθυνος για τον υπολογισμό της ευκλείδειας απόστασης του διανύσματος από ένα διάνυσμα υποστήριξης υψωμένης στο τετράγωνο. Σε κάθε επανάληψη υπολογίζεται η διαφορά μεταξύ των αντίστοιχων χαρακτηριστικών των δύο διανυσμάτων και υψώνεται στο τετράγωνο. Αντί να υπολογίζεται κάθε φορά μία μόνο διαφορά θα μπορούσαν να υπολογίζονται περισσότερες και να αθροίζονται σταδιακά σε μια μεταβλητή η οποία στο τέλος του βρόχου θα περιέχει την τετραγωνισμένη νόρμα. Η άθροιση όμως πολλών αριθμών κινητής υποδιαστολής συνεπάγεται μεγάλο κρίσιμο μονοπάτι επειδή οι προσθέσεις γίνονται σειριακά αν και δεν υπάρχει εξάρτηση μεταξύ των προσθετέων. Η πρόσθεση μπορεί να υλοποιηθεί αποδοτικά αν χρησιμοποιηθεί μια δενδρική μορφή. Ο εσωτερικός βρόχος εκτυλίσσεται τόσες φορές όσες διαφορές θα υπολογιστούν ταυτόχρονα. Οι διαφορές υπολογίζονται παράλληλα μεταξύ τους όπως και οι υψώσεις των διαφορών στο τετράγωνο. Στη συνέχεια οι διαθέσιμες τιμές προστίθενται ανά δύο και τα αποτελέσματα κρατώνται σε προσωρινές μεταβλητές. Αυτές προστίθενται και πάλι ανά δύο κ.ο.κ. μέχρι τον υπολογισμό της ολικής νόρμας στο τετράγωνο.

Η παραπάνω ιδέα υλοποιήθηκε για εκτύλιξη του εσωτερικού βρόχου 3,6 και 18 φορές που αντιστοιχεί σε πλήρη εκτύλιξη. Τα αποτελέσματα συγκεντρώνονται στ ΞΞΞ, όπου πραγματοποιείται σύγκριση μεταξύ εκτύλιξης του βρόχου με το χέρι, χρησιμοποιώντας δενδρική δομή και εκτύλιξης του βρόχου με τις αυτόματες τεχνικές του εργαλείου. Παρατηρείται σημαντική βελτίωση στο latency όταν η εκτύλιξη γίνεται με το χέρι και μάλιστα η διαφορά μεγαλώνει όσο μεγαλώνει και ο παράγοντας της εκτύλιξης. Η χρησιμοποίηση των DSP, LUTs και Flip Flop αυξάνεται καθώς η αντιγραφή του σώματος του εσωτερικού βρόχου οδηγεί στην δέσμευση περισσότερων πόρων προκειμένου οι πράξεις να δρομολογηθούν ταυτόχρονα.

Επίπεδο Βελτιστοποίησης 2: Διερεύνηση του Χώρου Σχεδιασμόυ των Αυτόματων HLS Τεχνικών Βελτιστοποίησης.

Η απόδοση μπορεί να βελτιωθεί περαιτέρω από το συνδυασμό των προηγούμενων τεχνικών με τις ενσωματωμένες αυτόματες τεχνικές βελτιστοποίησης που παρέχει το HLS και ονομάζονται directives. Η επιλογή αυτών εξαρτάται από την εγγενή παραλληλία του αλγορίθμου και τον τρόπο με τον οποίο αυτή μπορεί να αξιοποιηθεί. Στον συγκεκριμένο ταξινομητή επιλέγονται τεχνικές που στοχεύουν στην παραλληλοποίηση των βρόχων και της πρόσβασηα σε πίνακες στο εσωτερικό των βρόχων:

Pipeline: Αυτή η τεχνική εφαρμόζεται σε όλους τους βρόχους. Οι πράξεις των επαναλήψεων εκτελούνται παράλληλα κι όχι σειριακά χρησιμοποιώντας όλους τους πόρους κάθε χρονική στιγμή.

Εκτύλιξη βρόχου: Εφαρμόζεται σε όλους τους βρόχους. Δημιουργούνται αντίγραφα του σώματος του βρόχου ενώ μειώνεται ο αριθμός εκτελέσεων.

Διαίρεση Πίνακα: Εφαρμόζεται στους πίνακες sup_vector και sv_coef arrays. Διαιρεί τους πίνακες σε πίνακες μικρότερου μεγέθους κι άρα αυξάνεται ο αριθμός των θυρών ανάγνωσης. Η διαίρεση γίνεται κυκλικά (ανά κάποιο παράγοντα τα στοιχεία ανήκουν στην ίδια υποδιαίρεση πίνακα) ώστε να είναι δυνατή η ταυτόχρονη πρόσβαση σε διαδοχικά στοιχεία του αρχικού πίνακα με τη σειρά που αυτά χρειάζονται και στο βρόχο.

Μορφοποίηση Πίνακα: Εφαρμόζεται στους ίδιους πίνακες με την προηγούμενη τεχνική και για τον ίδιο σκοπό. Η διαφορά είναι ότι οι μικρότεροι πίνακες ενώνονται και πάλι σε έναν πίνακα ώστε ένα στοιχείου του νέου πίνακα να αποτελείται από όλα τα αντίστοιχα στοιχεία των μικρότερων πινάκων.

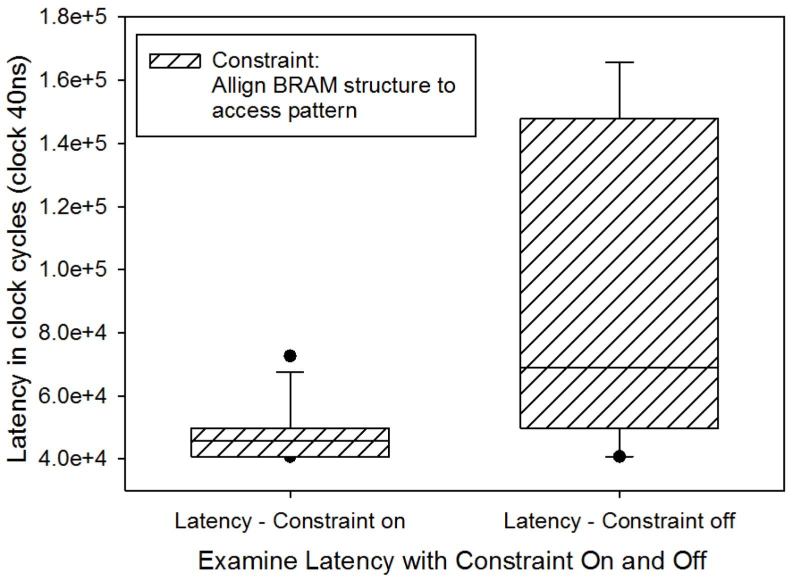
Η διερεύνηση όλων των συνδυασμών των επιλεγμένων τεχνικών οδηγεί σε έναν τεράστιο σχεδιαστικό χώρο και η εξαντλητική αξιολόγηση των λύσεων του δεν είναι εφικτή εντός εύλογου χρονικού διαστήματος. Προτείνουμε μια μεθοδολογία ώστε να έχουμε στη διάθεσή μας έναν μικρότερο σχεδιαστικό χώρο εντοπισμένο στις καλύτερες λύσεις του αρχικού εξαντλητικού χώρου. Αυτό επιτυγχάνεται μέσω τριών κανόνων μείωσης του χώρου διερεύνησης που στηρίζονται στην ιδέα ότι οι πιο αποδοτικές αρχιτεκτονικές είναι αυτές στις οποίες η ιεραρχία και διάταξη της μνήμης έχει προσαρμοστεί στο αλγοριθμικο μοτίβο πρόσβασης της μνήμης. Οι κανόνες είναι οι εξής:

Κανόνας 1. *Ο παράγοντας διαίρεσης ενός πίνακα πρέπει να ίσος με τον παράγοντα εκτύλιξης του βρόχου μέσα στον οποίο γίνεται η πρόσβαση στον πίνακα.*

Κανόνας 2. *Ο παράγοντας μορφοποίησης ενός πίνακα πρέπει να ίσος με τον παράγοντα εκτύλιξης του βρόχου μέσα στον οποίο γίνεται η πρόσβαση στον πίνακα.*

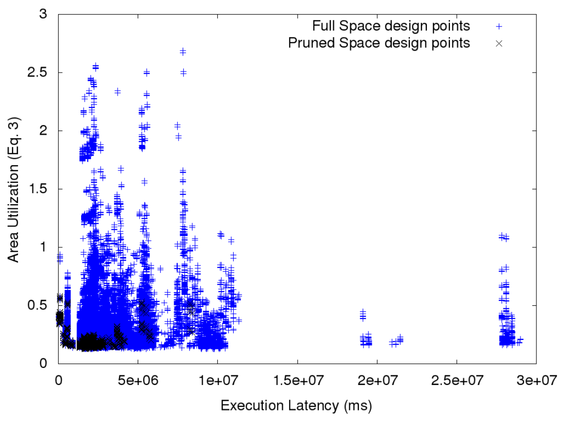
Κανόνας 3. *Το γινόμενο του παράγοντα μορφοποίησης και διαίρεσης ενός πίνακα πρέπει να ίσος με τον παράγοντα εκτύλιξης του βρόχου μέσα στον οποίο γίνεται η πρόσβαση στον πίνακα.*

Οι κανόνες αυτοί επιβεβαιώνονται πειραματικά αν απομονώσουμε μόνο τα configuration που συνδυάζουν τις παραπάνω τεχνικές και τα χωρίσουμε σε δύο υποσύνολα: το ένα υποσύνολο υπακούει στους κανόνες και το άλλο όχι. Είναι εμφανές από το Σχ.8.4 ότι οι κανόνες απομονώνουν τις πιο αποδοτικές λύσεις. Εφαρμόζοντας τους κανόνες σε όλο τον χώρο σχεδίασης προκύπτει ένας μικρότερος χώρος, ο *περικομμένος*. Απεικονίζουμε και τους δύο χώρους στο Σχ.8.5 και πραγματοποιούμε και στους δύο ανάλυση Pareto, η οποία παρέχει τα βέλτιστα σημεία ως προς τον χρόνο εκτέλεσης και χρησιμοποίησης πόρων. Το συμπέρασμα είναι ότι ο χώρος σχεδίασης μειώθηκε από 70962 λύσεις σε 2212, 3% του αρχικού χώρου. Ένα ποσοστό 33% των βέλτιστων σημείων κατά Pareto του αρχικού χώρου περιλαμβάνεται και στον περικομμένο.

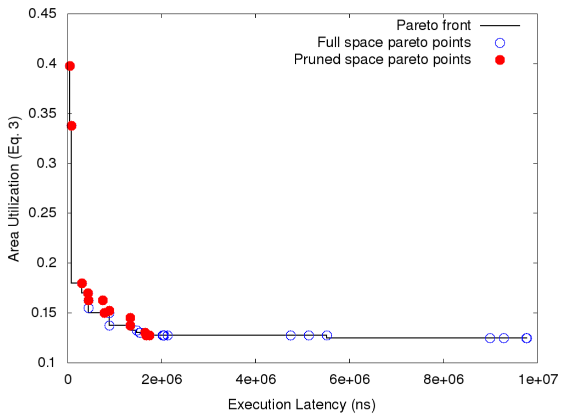


Σχήμα 8.4.: Επαλήθευση των κανόνων μείωσης του σχεδιαστικού χώρου.

Η μείωση του χώρου λύσεων μεταφράζεται σε μείωση του χρόνου αξιολόγησης των λύσεων. Εξαντλητική αξιολόγηση του περικομμένου χώρου είναι απαραίτητη προκειμένου να επιλεχθεί



(αʹ) Πλήρης και περικομμένος χώρος σχεδίασης.



(βʹ) Βέλτιστα σημεία κατά Pareto του πλήρους και περικομμένου χώρου σχεδίασης.

Σχήμα 8.5.: Σύγκριση του πλήρους και περικομμένου χώρου σχεδίασης.

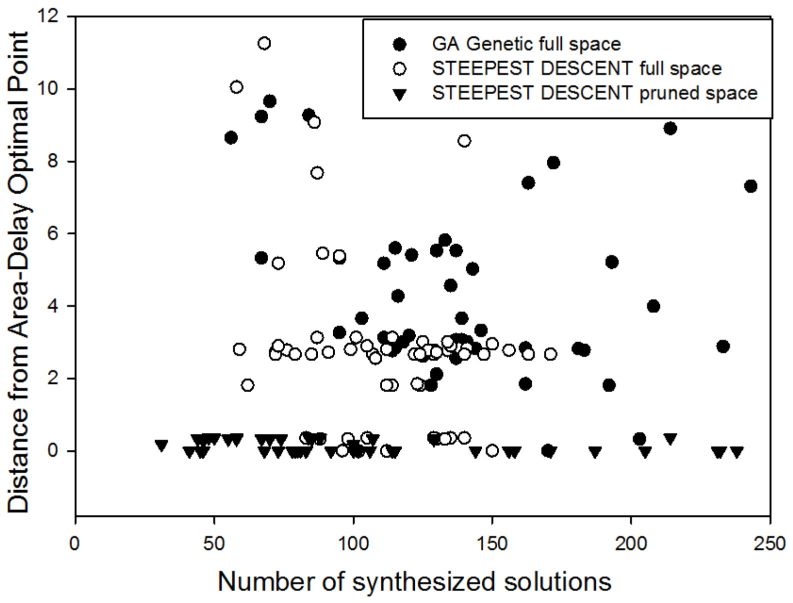
μία μεμονωμένη λύση. Αυτός ο χρόνος έχει μειωθεί από 15 μέρες σε 5 ώρες. Αν και εφικτό, στόχος μας είναι να προτείνουμε μια ενιαία μεθοδολογία που παραδίδει μία μόνο λύση στο χρήστη και τον απαλλάσει από το φόρτο χειρωνακτικής αξιολόγησης και εύρεσης λύσης. Για αυτό το σκοπό χρησιμοποιούμε έναν discrete steepest decent greedy optimizer ο οποίος ξεκινά από τυχαίες λύσεις του περικομμένου χώρου και καταλήγει σε μια λύση βελτιστοποιώντας την παρακάτω εξίσωση:

$$\min_{x \in \boldsymbol{D}} \left[\; Delay(\mathbf{x}) \times Area_{util}(\mathbf{x}) \; \right] \in R^2 \tag{8.2}$$

Οι τρεις διαφορετικές προσεγγίσεις διερεύνησης του χώρου λύσεων απεικονίζονται στο Σχ.;;.

Σε αυτό το σημείο αξιολογούμε την αποτελεσματικότητα της προτεινόμενης στρατηγικής εξερεύνησης σχεδιασμού για βελτιστοποίηση των SVM σε σύγκριση με την εξαντλητική διέλευση του αρχικού σχεδιαστικού χώρου. Για να ποσοτικοποιήσουμε την αποτελεσματικότητα της προτεινόμενης στρατηγικής εξερεύνησης, χρησιμοποιούμε δύο μετα-ευρετικές μεθόδους βελτιστοποίησης για την αναζήτηση σημείων σχεδίασης εντός του πλήρους και του περικομμένου σχεδιαστικού χώρου. Αυτά τα δύο μετα-ευρετικά βελτιστοποίησης είναι: 1) steepest descent και 2) ένας γενετικός optimizer με πληθυσμό 20 και 4 γενιές. Συγκρίνουμε τις τρεις εναλλακτικές λύσεις εξερεύνησης ως προς ι) τη βελτιστοποίηση των αποτελεσμάτων μέσω της μετρικής απόστασης σε σχέση με τις βέλτιστες λύσεις (όσο χαμηλότερη τόσο καλύτερα) που προκύπτουν από την εξαντλητική εξερεύνηση χώρου σχεδιασμού και (ιι) τον αριθμό των συνθετικών λύσεων που υποδεικνύει τη διεξαγωγή της εξερεύνησης-αποτελεσματικότητα χρόνου. Το Σχ.8.6 δείχνει την απόσταση από το σημείο σχεδίασης βέλτιστης περιοχής καθυστέρησης, μεταβάλλοντας τον αριθμό των συνθετικών σχεδίων για κάθε στρατηγική εξερεύνησης.

Όπως φαίνεται, η αποτελεσματικότητα κάθε στρατηγικής εξερεύνησης οργανώνεται σε διαφορετικές ζώνες εμβέλειας. Είναι σαφές ότι η ζώνη της προτεινόμενης μεθοδολογίας κυριαρχεί σχεδόν πλήρως και στα δύο μετα-ευρετικά βελτιστοποίησης που εφαρμόζονται στις παραλλαγές πλήρους σχεδιαστικού χώρου. Για τον ίδιο ή μικρότερο αριθμό configuration, η προτεινόμενη εξερεύνηση παρέχει σχεδιαστικές λύσεις που είναι πιο κοντά στα βέλτιστα σχέδια ΣΜ, παρέχοντας ένα μέσο σφάλμα απόστασης 0,001 με τυπική απόκλιση 0,14. Οι αντίστοιχες μέσες τιμές απόστασης για την πιο απότομη κάθοδο στον πλήρη χώρο σχεδιασμού έχουν υπολογιστεί σε 2,83 (τυπική απόκλιση: 2,37), ενώ για το γενετικό βελτιστοποιητή σε 4 (τυπική απόκλιση: 2,47). Είναι σημαντικό να τονίσουμε ότι η στρατηγική εξερεύνησης δύο φάσεων που παρουσιάζεται και επικυρώνεται εδώ είναι επιτυχής, επειδή ο βελτιστοποιητής μπορεί να αναζητήσει σημεία σχεδίασης σε έναν εξαιρετικά συμπαγή χώρο που περιλαμβάνει πολύ αποτελεσματικές λύσεις.



Σχήμα 8.6.: Μέση απόσταση από βέλτιστα σημεία για διαφορετικούς βελτιστοποιητές.

Τέλος, αξιολογούμε την αποτελεσματικότητα του βελτιστοποιημένου HW SVM χρησιμοποιώντας μια συν-σχεδιασμένη έκδοση HW/SW της εφαρμογής ανίχνευσης αρρυθμίας που βασίζεται στο ΗΚΓ. Το κομμάτι της ταξινόμησης υλοποιείται σε διαφορετική πλατφόρμα HW και τα αποτελέσματα συγκρίνονται μεταξύ τους. Η καθυστέρηση επικοινωνίας της παροχής νέων δεδομένων εισόδου στον ταξινομητή είναι αμελητέα (που κυμαίνεται από 10 έως 900 φορές λιγότερο από τον χρόνο υπολογισμού) αφού το διάνυσμα χαρακτηριστικών εισόδου αποτελείται από μόνο 18 σημεία. Όλα τα συστήματα λειτουργούν πάνω από ένα λειτουργικό σύστημα που βασίζεται σε Λινυξ και έχουν μεταγλωττιστεί χρησιμοποιώντας σημαίες O3 της gcc. Πιο συγκεκριμένα οι πλατφόρμες στόχου που χρησιμοποιούνται είναι:

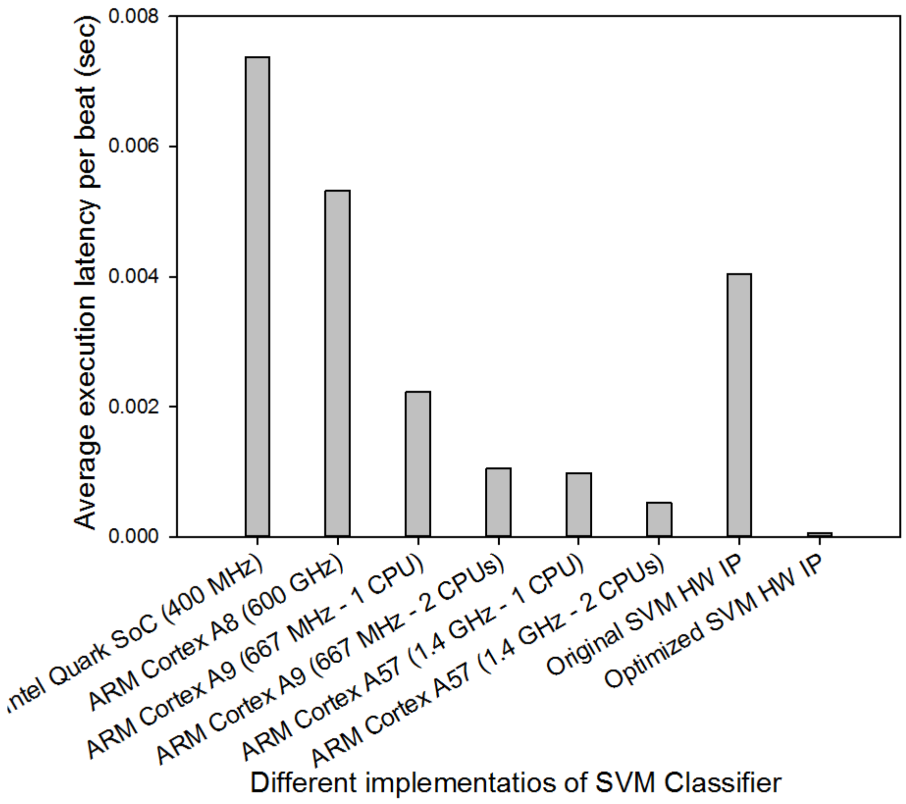
1. Intel Quark SoC 400 MHz.
2. ARM Cortex A8 600 MHz.
3. ARM Cortex A9 (Zynq Processing System) 667 MHz.
4. ARM Cortex A57 64-bit CPU 1.4 GHz.
5. Zedboard HW/SW co-designed system.

Το σετ δοκιμών αποτελείται από 52291 διανύσματα δοκιμής, τα οποία αντιστοιχούν σε διανύσματα που εξάγονται από καρδιακούς παλμούς των σημάτων ΗΚΓ. Αυτά παρέχονται ως είσοδος στις διαφορετικές υλοποιήσεις ταξινομητή SVM και η μέση καθυστέρηση εκτέλεσης του σταδίου διάγνωσης για όλες τις διαφορετικές υλοποιήσεις του ταξινομητή παρουσιάζεται στο Σχ.8.7. Παρατηρούμε μια συσχέτιση μεταξύ της ικανότητας της ;; και της μειωμένης καθυστέρησης εκτέλεσης. Η απλή HLS υλοποίηση στις περισσότερες περιπτώσεις δεν είναι αποτελεσματική ακόμη και σε σύγκριση με τις CPU. Αντίθετα, η *η βελτιστοποιημένη έκδοση του HW IP είναι σε κάθε περίπτωση πολύ πιο αποτελεσματική σε σύγκριση με όλες τις άλλες εναλλακτικές σχεδιάσεις.* Η επιτυγχανόμενη επιτάχυνση σε σύγκριση με την μη βελτιστοποιημένη έκδοση φτάνει έως και 78 φορές. Η βελτιστοποιημένη λύση είναι επίσης σχεδόν 10 φορές ταχύτερη σε σύγκριση με το σύστημα διπλού πυρήνα 64-bit ARM.

8.2. GANDAFL:Επιτάχυνση της αλληλούχισης μικρών τμημάτων DNA με χρήση dataflow HW επιταχυντή

Η ενότητα αυτή βασίζεται στη δημοσίευση μας [218].

Οι τεχνολογίες της αλληλουχίας επόμενης γενιάς (NGS) έχουν φέρει επανάσταση στη μελέτη του γονιδιώματος μέσω της ταχείας παραγωγής δεδομένων γονιδιωματικής με χαμηλό κόστος. Οποιαδήποτε ανάλυση γονιδιώματος συνήθως ξεκινά με ευθυγράμμιση ανάγνωσης DNA, χρησιμοποιώντας αλγόριθμους αντιστοίχισης συμβολοσειρών όπως ο Smith-Waterman για τη σύγκριση αλληλουχιών DNA. Η εγγενής υπολογιστική ένταση και η τεράστια ποσότητα δεδομένων εισόδου NGS που χειρίζεται ο αλγόριθμος καθυστερούν την ολοκλήρωση της εφαρμογής. Οι επαναπρογραμματιζόμενες υπολογιστικές συσκευές (FPGA) έχουν αξιοποιηθεί εκτενώς για την άμβλυνση αυτής της συμφόρησης, εστιάζοντας κυρίως σε υψηλής απόδοσης αν και αυτόνομες υλοποιήσεις, δηλαδή παραβλέποντας τις επιπτώσεις της ενσωμάτωσης των επιταχυνόμενων συναρτήσεων στα εργαλεία αλληλούχισης. Στις υπάρχουσες λύσεις επιτάχυνσης, ο αποτελεσματικός συν-σχεδιασμός της ευθυγράμμισης εξακολουθεί να παραμένει ανοιχτό ζήτημα, κυρίως λόγω της αμέλησης δυσκολιών της ενσωμάτωσης σε εργαλεία όπως το κόστος μεταφοράς δεδομένων μεταξύ λογισμικού

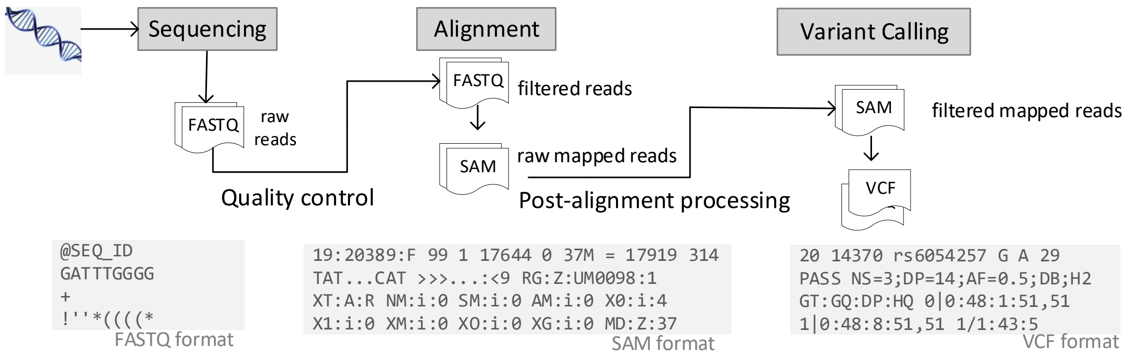


Σχήμα 8.7.: Μέσος χρόνος εκτέλεσης ανά παλμό.

και επιταχυντή και του κόστους κλήσης του επιταχυντή. Σε αυτό το κεφάλαιο, αντιμετωπίζουμε τις προαναφερθείσες αναποτελεσματικότητες και προτείνουμε τον GANDAFL, μια νέα αρχιτεκτονική για τα στάδια SmW Matrix-fill,Traceback της ευθυγράμμισης μικρών ακολουθιών DNA που προέρχονται από τεχνολογίες NGS. Στη συνέχεια, προτείνουμε μια ριζική αναδιάρθρωση του εργαλείου αλληλούχισης Βοωτιε2 που επιτρέπει την ευθυγράμμιση ανάγνωσης ομαδοποιημένων αλληλουχιών. Αυτή η αναδιάρθωση είναι καθοριστική για την αξιοποίηση της διαθέσιμης παραλληλίας και την επίτευξης της επιτάχυνσης. Η αλληλούχιση μεγάλων μπλοκ ακολουθιών αντί μεμονωμένων ελαχιστοποιεί την επιβάρυνση των κλήσεων των επιταχυντών, ενώ η μετακίνηση τόσο του Matrix-fill όσο και του Traceback στο τσιπ ελαχιστοποιεί το κόστος μεταφοράς δεδομένων. Ο επιταχυντής προσφέρει έως και $\times 116$ και $\times 2$ επιτάχυνση σε σχέση με σύγχρονους επιταχυντές λογισμικού και υλικού, αντίστοιχα, και ο Bowtie2 με ενσωματωμένο τον GANDAFL προσφέρει επιτάχυνση $\times 1{,}9$ σε σχέση με τον αρχικό.

Το Σχ.8.8 απεικονίζει μια τυπική ροή εντοπισμού παραλλαγών στο DNA. Το πρώτο βήμα εστιάζει στη δημιουργία των δεδομένων που απαιτούνται, τα οποία είναι μικρές ακολουθίες DNA και αποκαλούνται short reads. Τα μήκη τους κυμαίνονται από 50 έως 300 bp (ζεύγη βάσεων). Τα επόμενα βήματα, δηλαδή η αλληλούχιση και ο εντοπισμός παραλλαγών στο

DNA, διευκολύνουν την ανακατασκευή του γονιδιώματος του δείγματος και τη σύγκρισή του με το γονιδίωμα αναφοράς του οργανισμού. Η αλληλούχιση έχει ως στόχο να εντοπίσει μια θέση στο γονιδίωμα αναφοράς που είναι πιθανότατα η προέλευσή των short read. Τόσο οι αλληλουχίες αναφοράς όσο και οι ακολουθίες ανάγνωσης κωδικοποιούνται χρησιμοποιώντας αριθμητικές τιμές. Η αλληλούχιση έχει ως αποτέλεσμα ευθυγραμμίσεις που μπορούν είτε να ταιριάζουν τέλεια με την αντίστοιχη θέση αναφοράς είτε να περιλαμβάνουν αναντιστοιχίες. Συχνές παραλλαγές στην αλληλουχία των βάσεων που μπορεί δυνητικά να οδηγήσουν σε γενετικές μεταλλάξεις είναι οι εισαγωγές ή οι διαγραφές ή αντικαταστάσεις. Αυτές οι παραλλαγές ονομάζονται *edits*.



```
@SEQ_ID
GATTTGGGG
+
!''*((((*
```

```
19:20389:F 99 1 17644 0 37M = 17919 314
TAT...CAT >>>...:<9 RG:Z:UM0098:1
XT:A:R NM:i:0 SM:i:0 AM:i:0 X0:i:4
X1:i:0 XM:i:0 XO:i:0 XG:i:0 MD:Z:37
```

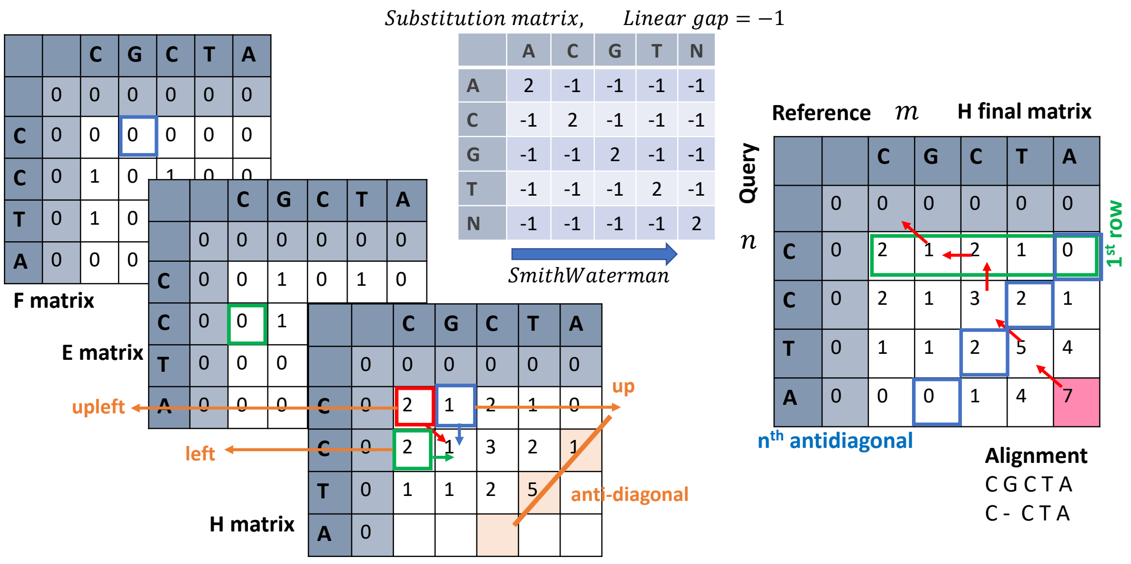
```
20 14370 rs6054257 G A 29
PASS NS=3;DP=14;AF=0.5;DB;H2
GT:GQ:DP:HQ 0|0:48:1:51,51
1|0:48:8:51,51 1/1:43:5
```

Σχήμα 8.8.: Τυπική ροή εργαλείων για εντοπισμό παραλλαγών στο DNA.

Σε αυτή τη δουλειά θα επικεντρωθούμε στο εργαλείο αλληλούχισης Bowtie2 το οποίο βασίζεται στο αλγοριθμικό μοντέλο λειτουργίας seed-and-extend. Σύμφωνα με αυτό, κάθε short read διαιρείται σε ακόμη μικρότερα τμήματα που λέγονται seeds για τα οποία εντοπίζονται αλληλουχίσεις στο γονιδίωμα αναφοράς που δεν εμφανίζουν καμία παραλλαγή. Κάθε ένα απο αυτά στη συνέχεια μπορεί να επεκταθεί σε μια πλήρη αλληλούχιση του αρχικού short read η οποία μπορεί να περιέχει και παραλλαγές. Κατά την αναζήτηση της πιο πιθανής προέλευσης, ελέγχονται με σειρά προτέραιοτητας τα seeds. Ωστόσο, η σειρά που ελέγχονται και το συνολικό πλήθος δοκιμών δεν είναι γνωστά εκ των προτέρων και διαφέρει για κάθε short read.

Το δεύτερο στάδιο της επέκτασης του seed γίνεται με χρήση του αλγορίθμου δυναμικού προγραμματισμού Smith-Waterman, ο οποίος εντοπίζει τις ομοιότητες μεταξλύ δύο ακολουθιών, του short read και του αντίστοιχου reference sequence. Αποτελείται από δύα στάδια. Το Matrix-Fill στάδιο συμπληρώνει έναν πίνακα ομοιότητας με σκορ με βάση τις ακόλουθες εξισώσεις:

$$
\begin{aligned}
E_{i,j} &= \max\{E_{i-1,j}, H_{i-1,j} - q\} - r \\
F_{i,j} &= \max\{F_{i,j-1}, H_{i,j-1} - q\} - r \\
H_{i,j} &= \max\{H_{i-1,j-1} + sc[Q[i], S[j]], E_{i,j}, F_{i,j}, 0\}
\end{aligned}
\tag{8.3}
$$

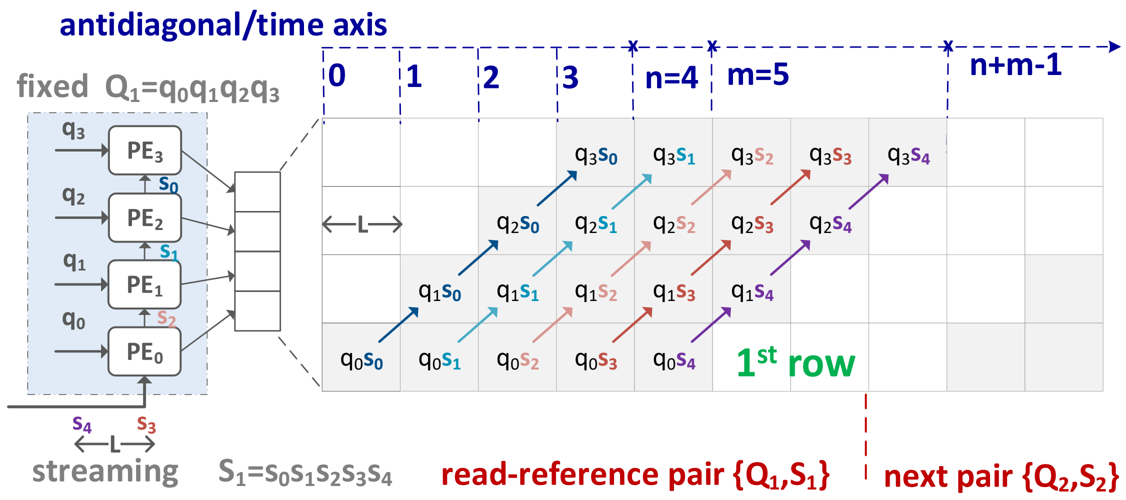


Σχήμα 8.9.: Εξαρτήσεις Δεδομένων στο Matrix Fill στάδιο και παράδειγμα Traceback για απλοποιημένο Smith-Waterman με γραμμικό μοντέλο για τα σκορ.

Όπως φαίνεται στο Σχ.8.9 η συμπλήρωση ενός κελιού εξαρτάται από το κελί στα αριστερά, από πάνω και διαγώνια. Αυτό σημαίνει ότι τα στοιχεία που ανήκουν στην ίδια διαγώνιο μπορούν να υπολογιστούν ταυτόχρονα και επομένως ο πίνακας να συμπληρωθεί σε τόσα βήματα όσες είναι και οι διαγώνιοι. Μόλις συμπληρωθεί, η Traceback ξεκινά από το κελί με το υψηλότερο σκορ και διατρέχει τον πίνακα ανάστροφα ώστε να φτιάξει την ακριβή αλληλούχιση με όλες τις δυνητικές παραλλαγές στις ακολουθίες. Η έξοδος της Traceback είναι το σκορ, η θέση της αλληλούχισης, οι θέσεις των παραλλαγών και ο τύπος τους.

Αρχιτεκτονική των Επιταχυντών

Στόχος αυτής της δουλειάς είναι η υλοποίηση FPGA επιταχυντή για τον Smith-Waterman. Υλοποιούμε δύο υπολογιστικές μονάδες, μία για κάθε στάδιο του αλγορίθμου. Οι δύο μονάδες δέχονται διαδοχικά ομάδες από short read προς αλληλούχιση. Η Matrix-Fill συμπληρώνει τον πίνακα για την πρώτη ομάδα και έπειτα στέλνει τα δεδομένα στην Traceback με αντίστροφη σειρά. Οι δύο μονάδες λειτουργούν σαν ένα pipeline. Η Matrix-Fill έχει υλοποιηθεί σαν ένας συστολικός πίνακας απο επεξεργαστικά στοιχεία. Καθένα από αυτά αναλαμβάνει να υπολογίσε μια γραμμή του πίνακα. Τα επεξεργαστικά στοιχεία λειτουργούν παράλληλα και μπορούν να υπολογίζουν έτσι παράλληλα κελιά του πίνακα τηρώντας τις εξαρτήσεις δεδομένων. Έτσι μπορούν και υπολογίζουν παράλληλα τα κελιά των διαγωνίων. Η εκμετάλλευση αυτής της παραλληλίας οδηγεί σε μια σειρά υπολογισμού των κελιών η οποία αντικατοπτρίζεται στη διάταξη του πίνακα στη μνήμη στον οποίο αποθηκεύονται οι τιμές. Αυτός ο πίνακας αντί να ακολουθεί τη διάταξη των στοιχείων ανά γραμμή, ακολουθεί διάταξη ανά διαγώνιο επομένως δημιουργείται ένα στρεβλομένο μοτίβο το οποίο απεικονίζεται

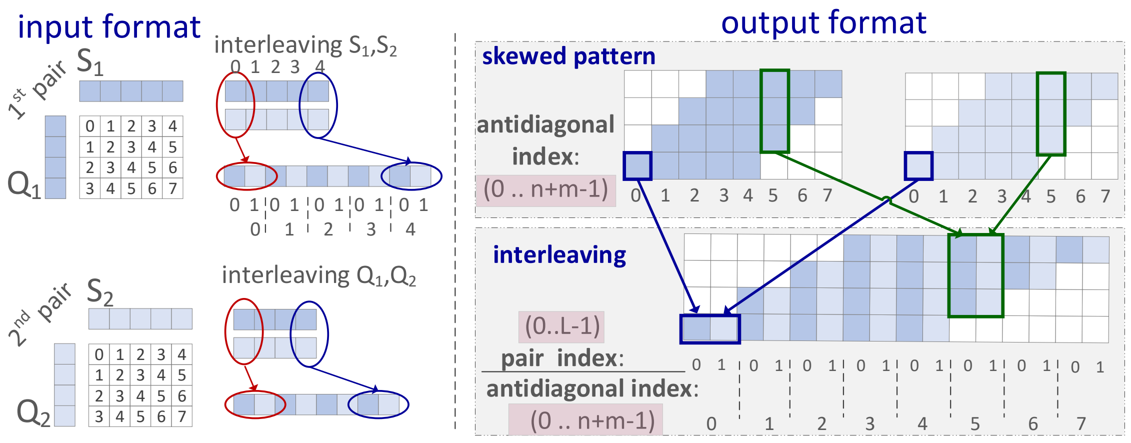


Σχήμα 8.10.: Παράδειγμα συμπλήρωσης του πίνακα H από τη συστοιχία PE στον άξονα του χρόνου για μήκη ακολουθιών $n = 4, m = 5$.

στο Σχ.8.10.

Η αρχιτεκτονική βελτιστοποιείται περαιτέρω με τη χρήση δύο τεχνικών. Η πρώτη αποσκοπεί στην εξάλειψη άεργων κύκλων ρολογιού στο εσωτερικό των επεξεργαστικών στοιχείων. Ο χρόνος υπολογισμού ενός κελιού αποτελείται από πολλούς κύκλους ρολογιού. Εμείς παρεμβάλλουμε δεδομένα από άλλα ζεύγη ακολουθιών προκειμένου να υπολογίζονται τα αντίστοιχα κελιά τα οποία δεν έχουν εξάρτηση δεδομένων από τους τρέχοντες υπογισμούς. Για αυτό το σκοπό οργανώνουμε την είσοδο τοποθετώντας στους ίδιους πίνακες εισόδου κυκλικά τα στοιχεία ακολουθιών διαφορετικών ζευγών. Αυτή η αναδιάρθρωση στοιχείων μεταφέρεται και στα ενδιάμεσα και τελικά αποτελέσματα όπως φαίνεται στο Σχ.8.11. Η δεύτερη τεχνική εξασφαλίζει ότι οι δύο μονάδες λειτουργούν ταυτόχρονα και σε pipeline, δηλαδή όσο η Matrix-Fill συμπληρώνει τους πίνακες μιας ομάδας ακολουθιών, η Traceback να μπορεί να διαβάζει τους πίνακες της προηγούμενης. Αυτό υλοποιείται με τη χρήση δύο αντιγράφων των πινάκων ώστε η Matrix-Fill να μην αναμένει την Traceback αλλά να προχωρά στην επόμενη ομάδα. Αυτή η τεχνική απεικονίζεται σχηματικά στο Σχ.8.12.

Διερευνούμε περαιτέρω την αποτελεσματικότητα της αρχιτεκτονικής αλλάζοντας τα μεγέθη εισόδου και υλοποιώντας πολλαπλά αντίτυπα του επιταχυντή στη συσκευή. Στην πρώτη περίπτωση, αυξάνονται τα μήκη των ακολουθιών. Αυτό έχει ως συνέπεια να αυξάνονται οι πόροι καθώς χρειάζονται τόσα επεξεργαστικά στοιχεία όσο είναι το μήκος της ακολουθίας. Ο χρόνος εκτέλεσης δεν αυξάνεται ωστόσο γραμμικά. Μεγάλη αύξηση του χρόνου εκτέλεσης παρατηρείται για μεγάλα μήκη ακολουθιών λόγω αυξημένης χρησιμοποίησης πόρων η οποία προκαλεί πτώση στο ρολόι. Εξετάζουμε επίσης το σενάριο αξιοποίησης όλων των διαθέσιμων συσκευών και πόρων και ελέγχουμε διάφορους συνδυασμούς αριθμού συσκευών και αριθμού επιταχυντών. Παρατηρούμε γραμμική επιτάχυνση εφόσον το ρολόι παραμένει υψηλό. Ο καλύτερος συνδυασμός προκύπτει με δύο επιταχυντές σε καθένα από τα δύο



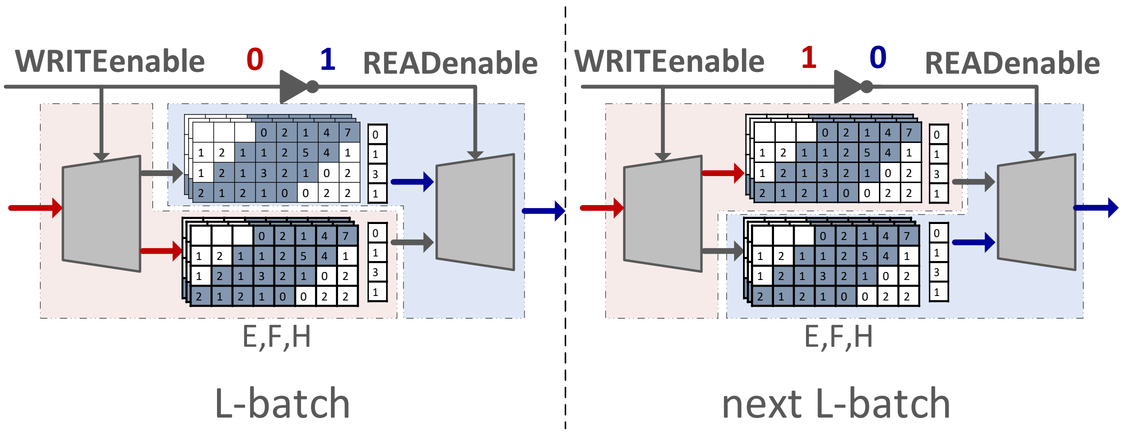
Σχήμα 8.11.: Παράδειγμα τεχνικής σύμπλεξης δεδομένων για L=2, όπως επηρεάζει τη δομή της μνήμης για δεδομένα εισόδου και εξόδου.

FPGA.

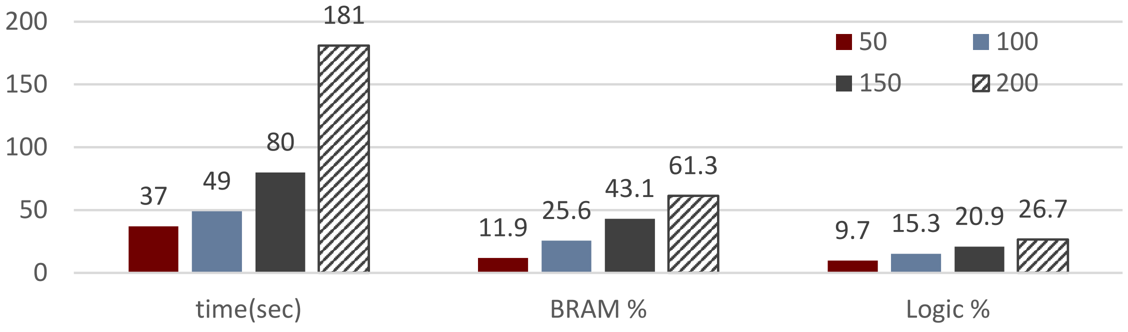
Αρχιτεκτονική του Ενσωματωμένου Συστήματος

Η ενσωμάτωση του επιταχυντή στο εργαλείο αλληλούχισης ενέχει κάποιους κινδύνους. Αυτοί πηγάζουν από το γεγονός ότι κάθε short read δημιουργεί διαφορετικό αριθμό από αλληλουχίσεις που πρέπει να ελεγχθούν ώστε να βρεθεί η πιο πιθανή τοποθεσία. Το Σχ.8.15α′ φανερώνει ότι αναλόγως την είσοδο, το Matrix-Fill μπορεί να απαιτεί το 48% του χρόνου αλλά αυτός ο χρόνος κατανέμεται σε πολλές προσπάθειες, 1 έως 170 για κάθε short read όπως μετρήθηκαν στο Σχ.8.15β′. Σε περίπτωση υλοποίησης μόνο αυτού ως επιταχυντή κι όχι όλου του SmW, αυτό ισοδυναμεί σε έως 500 εκατομμύρια μεταφορές των πινάκων που διαβάζει η Traceback κι άρα 186 terabytes. Η υλοποίηση της Traceback ως επιταχυντή, ελαχιστοποιεί αυτό το κόστος μεταφοράς σε 5 gigabytes. Οι προσπάθειες αλληλούχισης όμως παραμένουν πολλές και έχουν σαν αποτέλεσμα την κλήση των επιταχυντών εκατομμύρια φορές το οποίο επιβραδύνει την εκτέλεση έως 1000 φορές. Προκειμένου να αποφευχθεί αυτό, πρέπει να ομαδοποιηθούν οι αλληλουχίσεις και να εκτελούνται με λιγότερες κλήσεις στον επιταχυντή.

Αυτό επιτυγχάνεται αναγκαστικά με αναδιάρθρωση του κώδικα. Χωρίζουμε τον αλγόριθμο σε 3 φάσεις οι οποίες επαναλαμβάνονται κυκλικά μέχρι να εξαντληθούν/αλληλουχιστούν όλες οι ακολουθίες εισόδου. Κάθε εκτέλεση των 3 φάσεων αλληλουχεί μια ομάδα από ακολουθίες και καλεί τον επιταχυντή μία φορά. Στην πρώτη φάση κάθε εκτέλεσης βρίσκουμε τις πιθανές θέσεις αλληλούχισης για όλες τις ακολουθίες της ομάδας. Επιτρέπουμε έως 8 υποψηφίους για κάθε ακολουθία και διατάσουμε τα δεδομένα εισόδου τηρώντας την τεχνική εναλλάξ τοποθέτησης τους. Στη δεύτερη φάση καλούμε τον επιταχυντή και γίνεται η αλληλούχιση αυτών των δεδομένων ενώ στην τρίτη φάση κατανέμουμε και καταγράφουμε τα αποτελέσματα. Το Σχ.8.16 απεικονίζει την προτεινόμενη οργάνωση του κώδι-



Σχήμα 8.12.: Εναλλαγή των δικαιωμάτων Γραφής και Ανάγνωσης στα δύο αντίγραφα των πινάκων βάσει της τεχνικής Διπλής Αποθήκευσης.



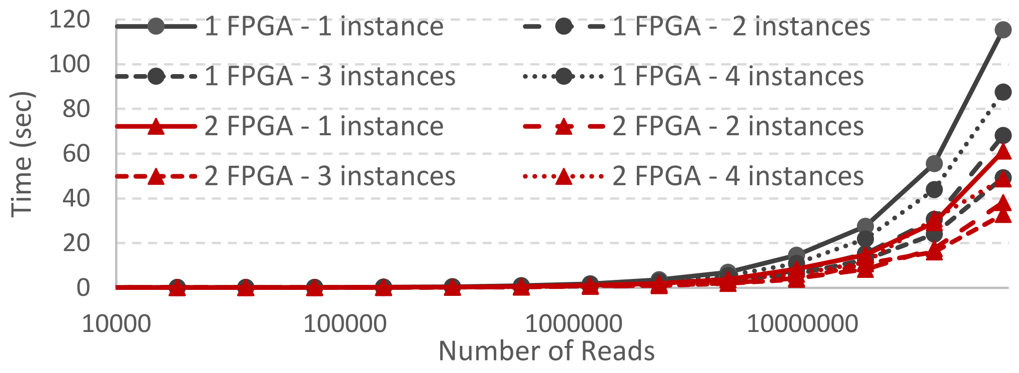
Σχήμα 8.13.: Χρόνος εκτέλεσης και χρησιμοποίηση πόρων για διάφορα μήκη ακολουθιών.

κα.

Πειραματική Αξιολόγηση του Επιταχυντή και του Συστήματος.

Η αξιολόγηση της προτεινόμενης αρχιτεκτονικής γίνεται και σε επίπεδο επιταχυντή αλλά και σε επίπεδο ενσωματωμένου συστήματος. Το υπολογιστικό σύστημα στο οποίο εκτελούμε τα πειράματα αποτελείται από έναν επεξεργαστή Intel Xeon E5-2658A που συνδέεται μέσω PCIe Gen 2 με την πλατφόρμα Maxeler's MAX5 που περιέχει δύο Xilinx VU9P Ultrascale FPGA. Τα δεδομένου εισόδου που χρησιμοποιούμε είναι: (1) τρία σύνολα που έχουν παραχθεί μέσω προσομοιωτή NEAT50,100,150 και (2) ένα σύνολο από πραγματικούς ασθενείς που νοσούν από CLL.

Αρχικά συγκρίνουμε τον επιταχυντή με άλλα εργαλεία αλληλούχισης υλοποιημένα σε software ή harware. Συγκεκριμένα συγκρίνουμε με τα εργαλεία: Edlib [114],WFA [115],KSW2 [116],Bowtie2 SIMD SmW [113],software GenASM [136],open-source Maxeler SmW [235] και Darwin GACT [36]. Η σύγκριση γίνεται για τρία διαφορετικά μήκη ακολουθιών και όλες

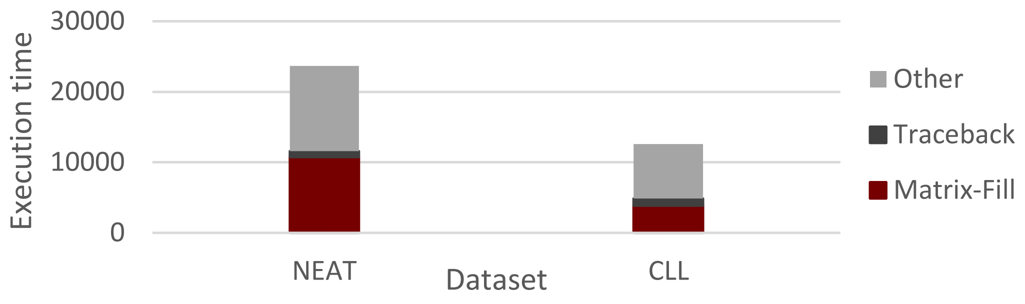


Σχήμα 8.14.: Απόδοση για διαφορετικό αριθμό επιταχυντών και συσκευών και αυξανόμενο μέγεθος εισόδου.

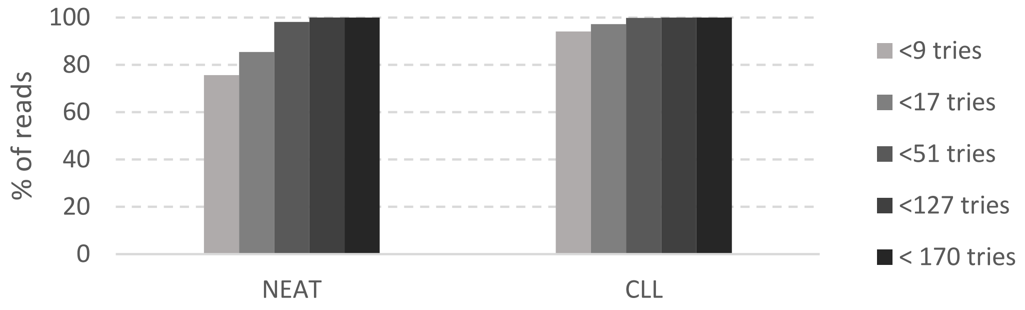
οι μετρήσεις σε software και hardware είναι σε τεχνολογία 18nm. Η υλοποίηση μας έχει 98.42 φορές μεγαλύτερο througphut σε σχέση με την ταχύτερη software υλοποίηση, δηλαδή την Edlib και 2.13 φορές από το ταχύτερο GACT hardware εργαλείο αλληλούχισης όπως φαίνεται στο Σχ.8.17α΄. Για δίκαιη σύγκριση, επαναλάβαμε το πείραμα μόνο για τις hardware αρχιτεκτονικές χρησιμοποιώντας το 80% των διαθέσιμων πόρων του ίδιου FPGA με πολλαπλές μονάδες του κάθε εργαλείου αλληλούχισης. Και πάλι, όπως παρουσιάζεται στο Σχ.8.17, η προτεινόμενη υλοποίηση έχει μεγαλύτερο throughput.

Κατόπιν ελέγχουμε την επιτάχυνση όταν ενσωματώνουμε τον επιταχυντή στον Bowtie2. Αν και περιορίζουμε τις υποψήφιες αλληλουχίσεις για κάθε ακολουθία σε 8, η ακρίβεια των αποτελεσμάτων έχει ελάχιστη απώλεια καθώς στην πλειοψηφία των ακολουθιών οι πρώτες δοκιμές είναι και οι επικρατούσες. Η επιτάχυνση επίσης φτάνει το θεωρητικό βέλτιστο, 1.92 αντί 1.96 φορές, για ένα ρεαλιστικό σύνολο εισόδου. Αγγίζει μόνο το 1.26 με θεωρητικά βέλτιστο το 1.67 για ένα σύνολο για το οποίο η υλοποίησή μας κάνει υπερεκτίμηση του αριθμού των δοκιμών που απαιτούνται και έχει αυξημένο κόστος χειρισμού και αλληλούχισης αυτών των δεδομένων (3 φορές περισσότερα δεδομένα). Τα ακριβή στατιστικά και το κέρδος φαίνονται στο Σχ.8.18.

Η προηγούμενη σύγκριση αφορά την περίπτωση που έχουμε ένα CPU thread στον Bowtie2. Ωστόσο το εργαλείο αυτό υποστηρίζει και εκτέλεση με πολλαπλά thread. Το σύστημά όμως διαθέτει πρόσβαση σε περιορισμένο πλήθος FPGA. Επομένως στο σενάριο εκτέλεσης που εξετάζουμε πολλαπλά thread θα πρέπει να διαμοιράζονται τους επιταχυντές. Χρησιμοποιούμε και τα δύο FPGA και σε καθένα από αυτά τοποθετούμε έναν επιταχυντή. Παρατηρούμε ότι η εκδοχή με τη χρήση επιταχυντών είναι πάντα ταχύτερη από την απλή και μάλιστα το κέρδος στο χρόνο εκτέλεσης φτάνει και τις 3 φορές για το μεγαλύτερο αριθμό thread που υποστηρίζει το σύστημά μας. Τα αποτελέσματα για αυξανόμενο αριθμό thread απεικονίζονται στο Σχ.8.19.



(α΄) Διαμερισμός του χρόνου εκτέλεσης κάθε λειτουργίας εντός του Bowtie2.



(β΄) Κατανομή των δοκιμών αλληλούχισης για κάθε ακολουθία των χρησιμοποιούμενων δεδομένων εισόδου.

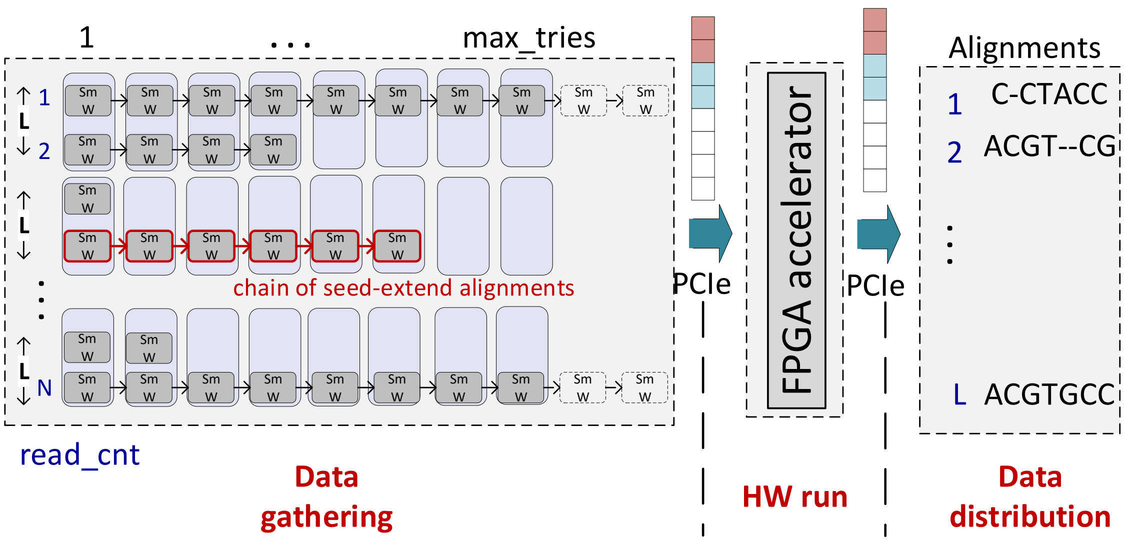
Σχήμα 8.15.: Μελέτη του Bowtie2 για διαφορετικά δεδομένα εισόδου.

8.3. GANDAFL:Επιτάχυνση του αλγορίθμου Banded Smith-Waterman για την αλληλούχιση μικρών τμημάτων DNA βασιζόμενη στο προφίλ των δεδομένων εισόδου

Η δουλειά αυτή παρουσιάζεται σε άρθρο μας που έχει γίνει δεκτό για δημοσίευση στο συνέδριο Design Automation Conference (DAC) 2023.

Σε αυτό το κομμάτι εξετάζουμε μια εναλλακτική προσέγγιση, η οποία συνδυάζει μια ευριστική υλοποίηση του Smith-Waterman και ένα στάδιο φιλτραρίσματος των αρχικών δεδομένων. Μελέτη των δεδομένων εισόδου υποδεικνύει ότι η αλληλούχιση συνήθως είναι ακριβής και εντοπίζεται μικρός αριθμός διαφοροποιήσεων από το ανθρώπινο γονιδίωμα. Αυτό μειώνει το χώρο αναζήτησης των λύσεων και μας επιτρέπει να χρησιμοποιήσουμε τον ευριστικό Banded Smith Waterman ο οποίος επιτελεί την ίδια λειτουργία, εντοπίζει λιγότερες διαφοροποιήσεις και καταναλώνει λιγότερους πόρους στο υλικό. Προτείνουμε λοιπόν ένα σύστημα που πλέον αποτελείται από πολλούς επιταχυντές και καλύπτει έως έναν αριθμό διαφοροποιήσεων ενώ εντοπίζει πλέον τις αλληλουχίσεις με ταχύτερο ρυθμό.

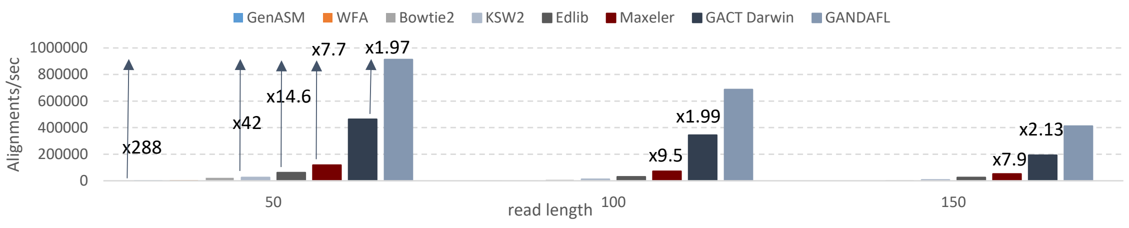
Ιδέα και Αρχιτεκτονική του Συστήματος



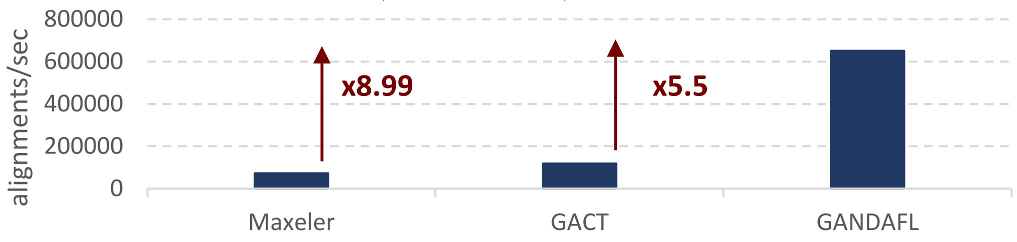
Σχήμα 8.16.: Αναδιάρθρωση του κώδικα του Bowtie2.

Για να αξιολογήσουμε την επίδραση της επεξεργασίας του προφίλ εισόδου στην αρχιτεκτονική, μελετούμε προσεκτικά τη συμπεριφορά και τα αποτελέσματα κατά την αλληλούχιση τριών διαφορετικών συνόλων δεδομένων εισόδου: ένα προσομοιωμένο σύνολο δεδομένων από 60 εκατομμύρια δείγματα μήκους 100 βάσεων που δημιουργήθηκαν από το προσομοιωτή NEAT , 60 εκατομμύρια δείγματα από το συχνά χρησιμοποιούμενο δείγμα NA12878 και το CLL, ένα πραγματικό σύνολο δεδομένων από ασθενείς που πάσχουν από χρόνια λεμφοκυτταρική λευχαιμία. Η Εικ. 8.20 παρουσιάζει ένα ιστόγραμμα του αριθμού των edits για όλες τις αλληλουχίσεις που βρέθηκαν για κάθε σύνολο δεδομένων. Οι επιλεγμένες τιμές ορίου edits στα ιστογράμματα, δηλαδή 2, 5, 10, 18, 30 , αντιστοιχούν σε κρίσιμες τιμές για τις οποίες σημειώνεται σημαντική αύξηση του αριθμού των ευθυγραμμίσεων. Για το προσομοιωμένο σύνολο δεδομένων, το 70.8% των εξετασθέντων ευθυγραμμίσεων περιλαμβάνει το πολύ ένα edit. Το ίδιο ποσοστό είναι ίσο με 74.4% και 82% για τα σύνολα δεδομένων NA12878 και CLL. Ο αριθμός των αναγνώσεων που αλληλουχίζονται επιτυχώς αυξάνεται γρήγορα (κατά 5-10%) για edits που κυμαίνονται από 1 έως 8. Ωστόσο, για τιμές μεγαλύτερες από 10, οι αυξητικές αλλαγές είναι της τάξης 0.01%. Πράγματι, για όλα τα σύνολα δεδομένων, το 99.99% των ευθυγραμμίσεων περιέχει λιγότερες από 18 edits.

Το πρώτο σημαντικό εύρημα της διερεύνησης είναι η πιθανή επίδραση του χαμηλού αριθμού επεξεργασιών στην αρχιτεκτονική του επιταχυντή. Η Εικ. 8.21 απεικονίζει την τοποθέτηση μιας ακριβούς ευθυγράμμισης καθώς και μιας ευθυγράμμισης με edits. Και οι δύο ευθυγραμμίσεις ξεκινούν από το seed hit, δηλαδή το πρώτο σημείο αντιστοίχισης, ωστόσο η ευθυγράμμιση με edits αποκλίνει από την ¨κύρια διαγώνιο¨. Όσο λιγότερα είναι τα edit, τόσο στενότερη είναι η ζώνη του πίνακα που καλύπτει η ευθυγράμμιση. Αυτό αξιοποιείται από τον Banded Smith-Waterman [247], που βασίζεται στην παρατήρηση ότι το μέγιστο σκορ στον πίνακα εμφανίζεται γύρω από την ¨κύρια διαγώνιο' σε μέγιστη απόσταση ίση με τον αριθμό



(α΄) Throughput σύγκριση για διαφορετικά μήκη ακολουθιών για τους GenASM, Edlib, WFA, KSW2, Bowtie2 SmW, Maxeler(only MatrixFill), GACT Darwin, GANDAFL.



(β΄) Chip-to-chip throughput σύγκριση για τα Maxeler(MatrixFill), GACT, GANDAFL.

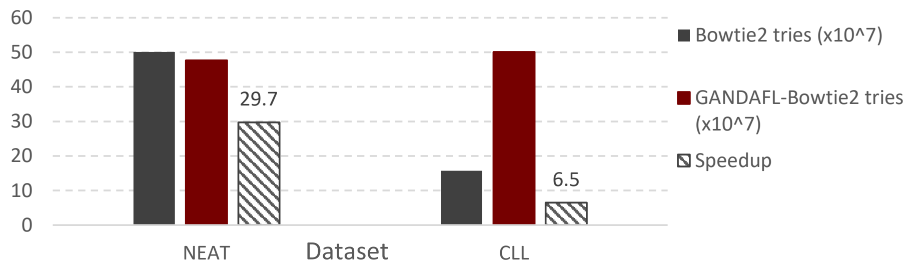
Σχήμα 8.17.: Σύγκριση του τηρουγηπυτ για διαφορετικά εργαλεία αλληλούχισης.

των edit.

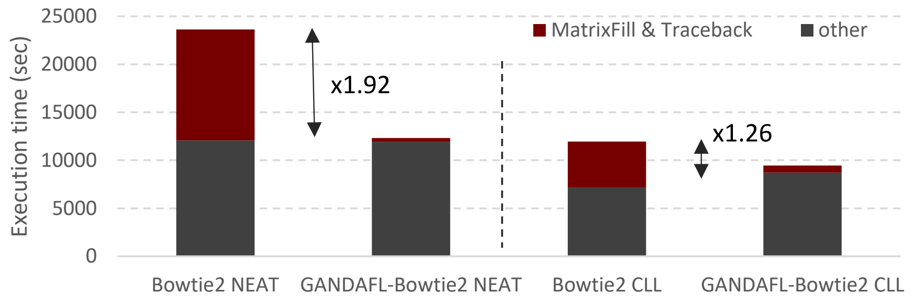
Επομένως, για ένα όριο edit, ο Banded Smith-Waterman επικεντρώνεται στην αντίστοιχη ζώνη εντός του πίνακα και εξαλείφει την ανάγκη αποθήκευσης και αναζήτησης ολόκληρων των πινάκων σκορ. Η Εικ. 8.21 δείχνει τη σχετική ζώνη εντός του πλήρους πίνακα, λαμβάνοντας υπόψη το όριο edit. Η τελική ευθυγράμμιση θα βρεθεί εντός της περιοχής που σημειώνεται από την ¨κύρια διαγώνιο¨, τις ανώτερες διαγωνίους και τις κατώτερες διαγωνίους γύρω από την ¨κύρια διαγώνιο' με όριο edit. Επομένως, η μελέτη του του συνόλου δεδομένων εισόδου μπορεί να μας βοηθήσει να αποφύγουμε την υπερβολική παροχή πόρων.

Το δεύτερο σημαντικό εύρημα δεν αφορά το χαμηλό όριο edit αυτό καθαυτό, αλλά τη συχνότητα με την οποία εμφανίζεται κάθε όριο. Η κατανομή ανά όριο edit μεταφράζεται σε αντίστοιχη κατανομή του χρόνου που απαιτείται από κάθε τύπο ευθυγράμμισης, δηλαδή οι ευθυγραμμίσεις με 0-1 edit καταλαμβάνουν τουλάχιστον το 70% του συνολικού χρόνου ευθυγράμμισης του συνόλου δεδομένων. Αυτή η ομοιογένεια υποδηλώνει ότι η συνολική απόδοση θα μπορούσε να επωφεληθεί περισσότερο εάν δημιουργήσουμε πολλαπλούς επιταχυντές μικρότερου μεγέθους και τους αναθέσουμε στους αντίστοιχους τύπους ευθυγράμμισης με μια παρόμοια κατανομή.

Αξιοποιούμε τα παραπάνω ευρήματα για να δημιουργήσουμε ένα σύστημα με πολλαπλούς επιταχυντές, δηλαδή ένα πολυεπεξεργαστικό σύστημα δεδομένων, προσαρμοσμένο στις προδιαγραφές του συνόλου δεδομένων όσον αφορά το όριο edit. Όπως φαίνεται στην Εικ. 8.22, αυτό μπορεί να επιτευχθεί αρχικά με τον προσδιορισμό του προφίλ του συνόλου δεδο-

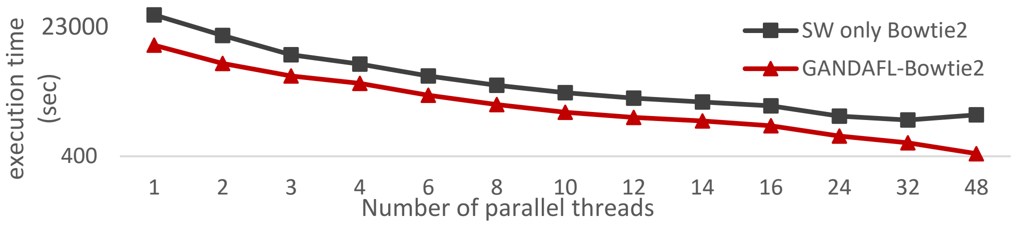


(α΄) Σύγκριση στατιστικών αλληλούχισης και χρόνου εκτέλεσης αλληλούχισης μεταξύ αρχικού Bowtie2 και GANDAFL-Bowtie2.



(β΄) Επιτευχθείσα επιτάχυνση του GANDAFL-Bowtie2.

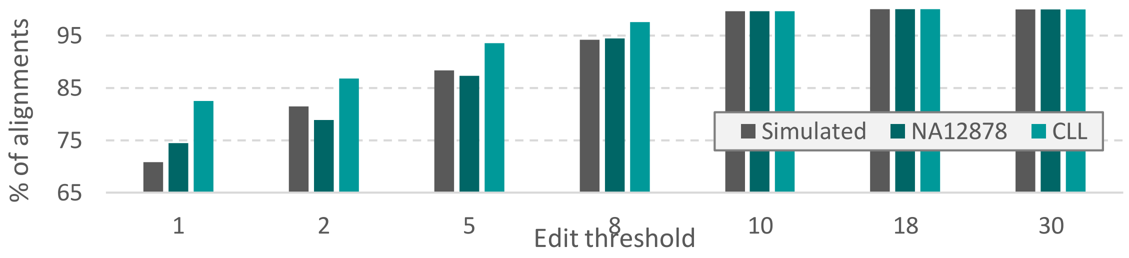
Σχήμα 8.18.: Αξιολόγηση του Bowtie2 με ενσωματωμένο τον προτεινόμενο επιταχυντή.



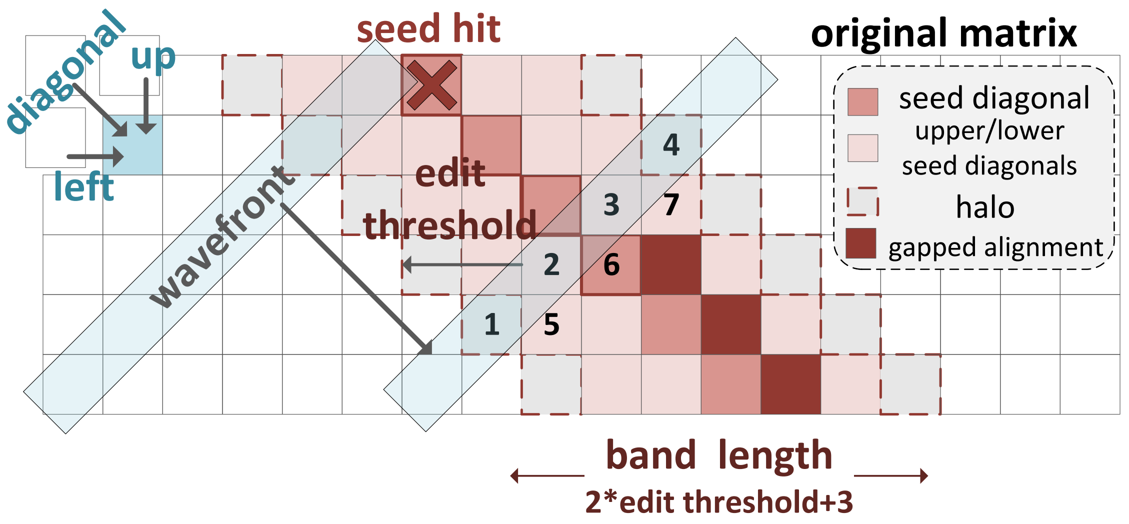
Σχήμα 8.19.: Σύγκριση της απόδοσης των συστημάτων για χρήση πολλών τηρεαδ και διαμοιρασμό δύο επιταχυντών.

μένων εισόδου για την εξαγωγή της κατανομής των edit και τον εντοπισμό των ορίων edit για τα οποία υπάρχει αισθητή άνοδος στον αριθμό των επιτυχών αλληλουχίσεων. Ο αριθμός των ορίων edit υποδηλώνει τον αριθμό των διαφορετικών τύπων επιταχυντών Banded Smith-Waterman προσαρμοσμένων στα edit. Κάθε τύπος μπορεί να υποστηρίξει έως ένα προκαθορισμένο αριθμό edit, ίσο με τα εντοπισμένα όρια. Ο αριθμός των επιταχυντών που εκχωρούνται για κάθε τύπο ακολουθεί την ίδια κατανομή με τα όρια επεξεργασίας. Στη συνέχεια, ο αλγόριθμος φιλτραρίσματος SneakySnake [119] κατηγοριοποιεί τις υποψήφιες ευθυγραμμίσεις εισόδου στην αντίστοιχη κατηγορία όριου edit και αναθέτει κάθε ευθυγράμμιση στον αντίστοιχο επιταχυντή.

Η προτεινόμενη αρχιτεκτονική πολυεπεξεργαστικής ροής δεδομένων περιλαμβάνει πολλούς μονάδες επιτάχυνσης Banded Smith-Waterman, οι οποίες έχουν αναπτυχθεί χρησιμοποι-



Σχήμα 8.20.: Κατανομή των edits για τις αλληλουχίσεις τριών διαφορετικών συνόλων δεδομένων.



Σχήμα 8.21.: Παράδειγμα Banded SmithWaterman για αριθμό edit 2.

ώντας την τεχνολογία της πολυεπεξεργαστικής ροής δεδομένων. Κάθε μονάδα ρυθμίζεται με ένα όριο edit που επιλέγεται κατά τη μελέτη των δεδομένων εισόδων με βάση τη μεθοδολογία που προτείνεται. Ο Banded Smith-Waterman περιλαμβάνει δύο υπομονάδες υπολογισμού, μία για κάθε στάδιο του, δηλαδή Matrix-Fill και Traceback. Η Matrix-Fill λαμβάνει ζευγάρια αναγνωσμάτων-αναφοράς ως υποψήφιες ευθυγραμμίσεις και υπολογίζει τουςς πίνακες E, F, H. Μόλις οι πίνακες είναι έτοιμοι, η Traceback τους διατρέχει ανάποδα για τον εντοπισμό των επεξεργασιών. Κάθε στιγμή, οι δύο υπομονάδες εκτελούνται παράλληλα και με τη μέθοδο του pipeline, δηλαδή καθώς η Matrix-Fill εκτελείται σε ένα ζευγάρι ευθυγραμμίσεων, η Traceback επεξεργάζεται τους πίνακες που δημιουργήθηκαν για το προηγούμενο ζευγάρι.

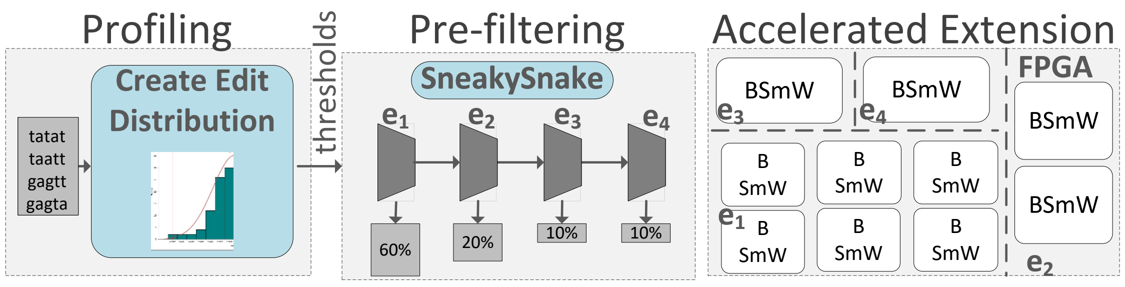
Η Εικόνα 8.23 απεικονίζει επίσης τον τρόπο με τον οποίο τα στοιχεία της ῾ζώνης᾿ αποθηκεύονται σε έναν μικρότερο πίνακα. Μόνο τα στοιχεία που αντιστοιχούν σε κελιά εντός της ζώνης ενδιαφέροντος αποθηκεύονται στην BRAM. Σημειώστε ότι η αποθήκευση των διαγωνίων σε μια BRAM αλλάζει τη σχετική θέση ορισμένων γειτονικών κελιών.

Πειραματική Αξιολόγηση του Επιταχυντή και του Συστήματος.

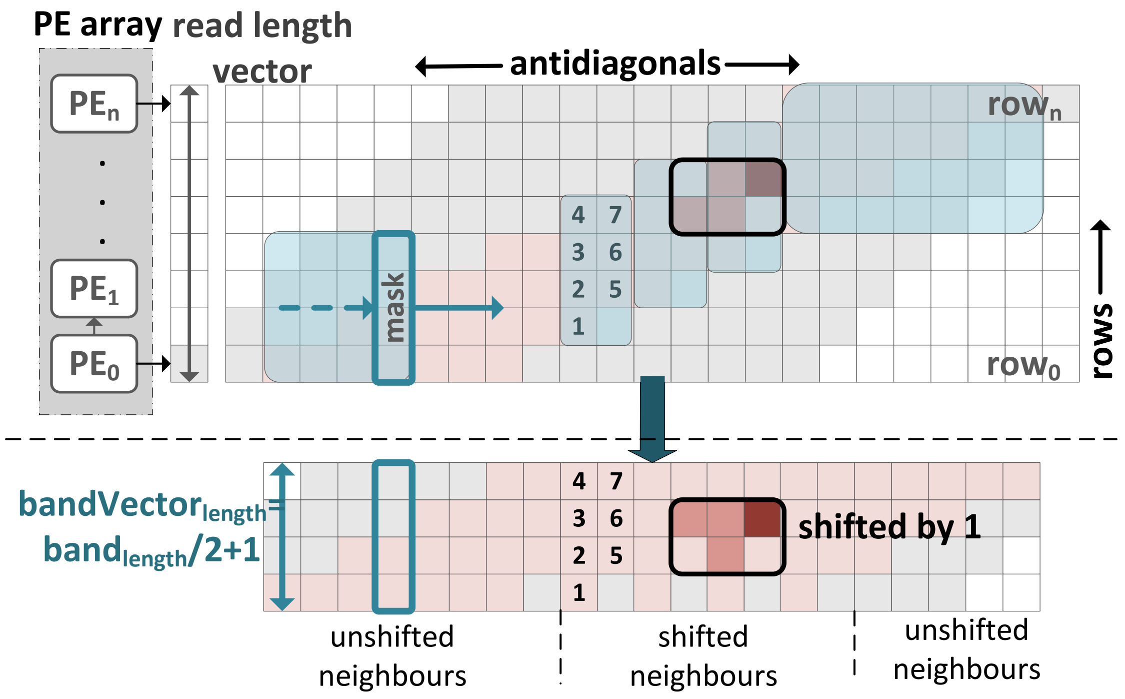
Σε αυτήν την ενότητα αξιολογείται η ακρίβεια της αλληλούχισης σύντομων αναγνώσματος κατά την εφαρμογή φιλτραρίσματος και αλληλούχισης μέσω της προτεινόμενης υλοποίησης Banded SmithWaterman για το αντίστοιχο όριο edit. Ως σύγκριση, χρησιμοποιούνται τα αποτελέσματα του ευθυγραμμιστή Bowtie2. Η προ-διαλογή υλοποιείται από το SneakySnake, το οποίο απορρίπτει τους υποψήφιους ευθυγραμμισμούς με περισσότερα edit από το καθορισμένο όριο edit. Λαμβάνονται υπόψη όρια edit που κυμαίνονται από 1 έως σχεδόν το 20% του μήκους του αναγνώσματος (100). Η σύγκριση της ακρίβειας παρουσιάζεται στο Σχήμα 8.24 για 3 σύνολα δεδομένων. Τα αποτελέσματα είναι σύμφωνα με τα ευρήματα της κατανομής των επεξεργασιών, καθώς για τα περισσότερα σύνολα δεδομένων ο ρυθμός ευθυγράμμισης συγκλίνει στον βέλτιστο όταν το όριο edit είναι μεγαλύτερο ή ίσο του 10.

Επίσης, συγκρίνουμε τον σχεδιασμό μας με δύο προηγμένους επιταχυντές υλικού, το GACT [36] και το GANDAFL [246]. Εξετάζουμε δύο διαφορετικά σενάρια για μια δίκαιη σύγκριση. Το πρώτο σενάριο συγκρίνει το ρυθμό αλληλούχισης μιας μονάδας του κάθε επιταχυντή, για να αξιολογήσουμε τη δυνατότητα του σχεδιασμού να παράγει μια αλληλούχιση ταχύτερα από τις προηγμένες λύσεις. Το δεύτερο σενάριο συγκρίνει σχεδιασμούς με πολλαπλούς επιταχυντές που αξιοποιούν τη μέγιστη χωρητικότητα της ίδιας συσκευής FPGA. Όπως φαίνεται στο Σχήμα 8.25, ο προτεινόμενος υλοποιημένος σχεδιασμός επιτυγχάνει μια επιτάχυνση $\times 1.53$ σε σχέση με τον GANDAFL dataflow και μια επιτάχυνση $\times 3$ σε σχέση με τον GACT μονάδας επιταχυντή. Χάρη στην αποτελεσματική χρήση πόρων, ο προτεινόμενος σχεδιασμός πολυεπεξεργαστικής ροής δεδομένων χωράει 8 επιταχυντές στο FPGA και επιτυγχάνει μια επιτάχυνση $\times 4.77$ σε σχέση με τις 3 πολυεπεξεργαστικές ροές δεδομένων του επιταχυντή GANDAFL. Το μοντέλο υπολογισμού με πολλαπλές ροές δεδομένων και η αποτελεσματική χρήση πόρων εξασφαλίζουν μια επιτάχυνση $\times 26.35$ σε σχέση με έναν σχεδιασμό του GACT με 12 μονάδες.

Βασιζόμενοι στην προτεινόμενη μεθοδολογία για πολυεπεξεργαστικά συστήματα, αξιοποιούμε τα προφίλ επεξεργασίας και κατασκευάζουμε τους αντίστοιχους σχεδιασμούς πολλαπλής ροής δεδομένων για ένα ρολόι 250MHz. Ο πίνακας 8.2 περιλαμβάνει τα διαφορετικά con-



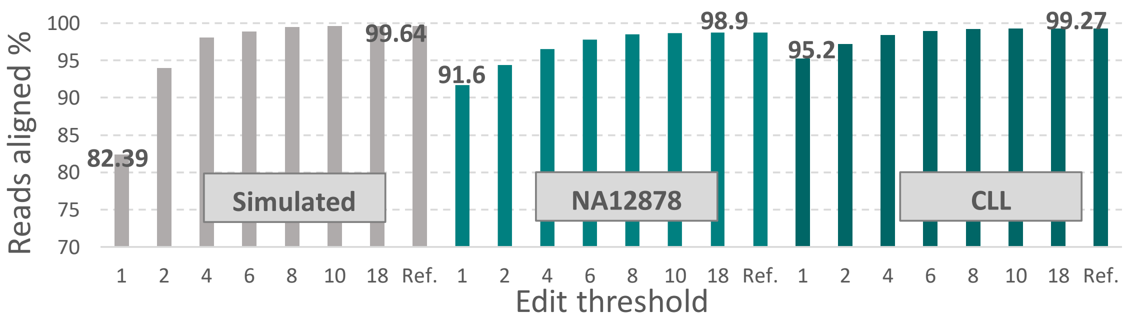
Σχήμα 8.22.: Επισκόπηση της στρατηγικής και αρχιτεκτονικής για επιτάχυνση βασισμένη σε ενδεικτική κατανομή edit της εισόδου σε ποσοστά 60-20-10-10%.



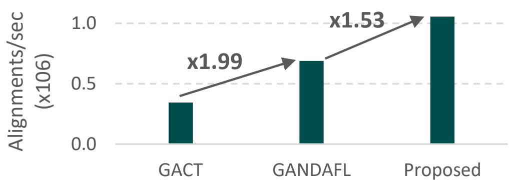
Σχήμα 8.23.: Υπολογισμός των πινάκων και βελτιστοποιημένο σχήμα χρησιμοποίησης πόρων μνήμης.

figuration για κάθε σύνολο δεδομένων. Συγκρίνουμε αυτούς τους σχεδιασμούς με τον GANDAFL, ο οποίος περιλαμβάνει 3 μονάδες που επιτρέπουν έως 17 edit και έχει ρολόι 200MHz.

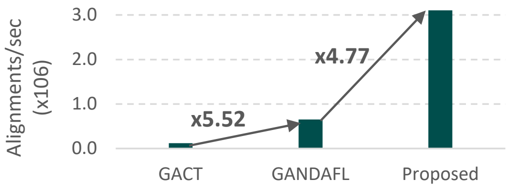
Και οι δύο σχεδιασμοί συγκρίνονται επίσης με τον ευθυγραμμιστή Bowtie2, ο οποίος δημιουργεί τις ευθυγραμμίσεις εισόδου και λειτουργεί ως αναφορά για τον ρυθμό ευθυγράμμισης. Ο αλγόριθμος προ-φιλτραρίσματος SneakySnake χρησιμοποιήθηκε για να ταξινομήσει τις υποψήφιες ευθυγραμμίσεις και να τις αναθέσει στον αντίστοιχο επιταχυντή ροής δεδομένων με βάση το όριο edit. Όπως φαίνεται στο Σχήμα 8.26, η προτεινόμενη προσέγγιση παρέχει επιτάχυνση έως και ×440 σε σχέση με τον Bowtie2 και ×1.8 σε σχέση με τη συμβατική προσέγγιση επιτάχυνσης του GANDAFL. Σημειώνεται ότι αυτή η επιτάχυνση δεν συνοδεύεται από μείωση της ακρίβειας ευθυγράμμισης, καθώς αυτός έχει συγκλίνει στον αναφορικό για τα επιλεγμένα edit.



Σχήμα 8.24.: Ακρίβεια αλληλούχισης με χρήση του SneakySnake φίλτρου και του προτεινόμενου Banded Smith-Waterman επιταχυντή.



(α΄) Σύγκριση throughput μεταξύ των επιταχυντών.

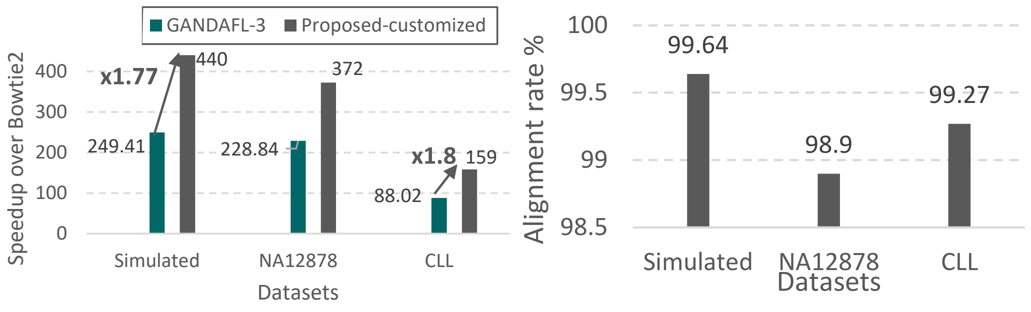


(β΄) Σύγκριση throughput μεταξύ πολλών μονάδων των επιταχυντών.

Σχήμα 8.25.: Σύγκριση throughput μεταξύ διαφορετικών HW επιταχυντών.

Πίνακας 8.2.: Multi-Dataflow Configurations customized to the edit profiles of input datasets.

Dataset	Configuration		
	Edit threshold	Instances	Alignments covered
Simulated	1-2-10-18	5-1-1-1	70.8-81.4-99.7-99.9%
NA127828	1-2-5-18	5-1-1-1	74.4-87.3-99.6-99.9%
CLL	1-5-18	6-1-1	82.5-93.51-99.9%



Σχήμα 8.26.: Αξιολογηση της απόδοσης και ακρίβειας του προτεινόμενου συστήματος πολυ-επεξεργαστών.

Γλωσσάρι

Behavioral Description	Περιγραφή κυκλώματος σε επίπεδο συμπεριφοράς.
Behavioral Synthesis	Σύνθεση κυκλώματος περιγραμμένο σε επίπεδο συμπεριφοράς.
Critical Path Delay	Καθυστέρηση κρίσιμου μονοπατιού του κυκλώματος.
Data Flow Graph (DFG)	Γράφος Ροής Δεδομένων.
Design Space Exploration (DSE)	Εξερεύνηση του χώρου σχεδίασης.
Dataflow computing	Προγραμματιστικό Μοντέλο περιγραφής αλγορίθμων. Προδιαγράφει την αρχιτεκτονική και επιμέρους μονάδες ενός συστήματος και στηρίζεται στη σωστή και συγχρονισμένη ροή δεδομένων μέσα από το hardware για την παραγωγή σωστών αποτελεσμάτων.
Deoxyribonucleic acid (DNA)	Το DNA είναι ένα μακρομόριο που αποτελεί το γενετικό υλικό όλων των οργανισμών. Διαμορφώνεται ως ένα μεγάλο μόριο που αποτελείται από δύο επιμήκεις αλυσίδες, οι οποίες συστρέφονται ελικοειδώς μεταξύ τους και συγκρατούνται μεταξύ τους με δεσμούς υδρογόνου. Κάθε αλυσίδα αποτελείται μια συστοιχία αζωτούχων βάσεων, συμπληρωματικών με τις βάσεις της άλλης αλυσίδας.
ECG	ΗΚΓ ή Ηλεκτροκαρδιογράφημα είναι μια απλή κλινική εξέταση η οποία καταγράφει την ηλεκτρική δραστηριότητα των μυών της καρδιάς και απεικονίζει στον άξονα του χρόνου τη μεταβολή του ηλεκτρικού δυναμικού.

Gate-level Description	Περιγραφή κυκλώματος σε Επίπεδο Λογικών Πυλών. Η περιγραφή αυτή μπορεί να παραχθεί με την σύνθεση της RTL περιγραφής ενός κυκλώματος για μια δεδομένη τεχνολογική βιβλιοθήκη.
Hardware Accelerator	Κύκλωμα ειδικού σκοπού/συνεπεξεργαστής υλικού που σχεδιάζεται/χρησιμοποιείται για την επιτάχυνση μιας εφαρμογής/διαδικασίας.
HLS	High Level Synthesis. Σύνθεση κυκλώματος από υψηλά επίπεδα σχεδιαστικής αφαίρεσης.
Linear Regression	Απλή γραμμική παλινδρόμηση.
Mean Error Distance (MED)	Μέση Απόλυτη Απόσταση. Είναι η μέση τιμή της απόλυτης απόστασης (ED) για την εξεταζόμενη κατανομή εισόδου και ορίζεται ως $MED = \frac{1}{M}\sum_{i=1}^{M}\lvert P_i - P_i'\rvert$, όπου P_i είναι το ορθό αποτέλεσμα, P_i' το προσεγγιστικό αποτέλεσμα, και M το πλήθος των αποτελεσμάτων.
Next Generation Sequencing (NGS)	Μέθοδος Αλληλούχισης DNA Νέας Γενιάς. Είναι μια τεχνολογία που χρησιμοποιείται για τον προσδιορισμό αλληλουχιών DNA και την ανίχνευση μεταλλάξεων.
Pareto front	Μέτωπο Pareto. Σε προβλήματα βελτιστοποίησης με πολλαπλά κριτήρια, οι Pareto βέλτιστες λύσεις είναι αυτές για τις όποιες η βελτιστοποίηση του ενός κριτηρίου μπορεί να επιτευχθεί μόνο αν χειροτερεύσει τουλάχιστον ένα άλλο κριτήριο.
Register Transfer Level (RTL) Description	Περιγραφή κυκλώματος σε Επίπεδο Μεταφοράς Καταχωρητών.
Seed hit	Σημείο στο γονιδίωμα αναφοράς στο οποίο τμήμα του short read ευθυγραμμίζεται χωρίς κανένα κενό
Short Reads	Αλληλουχία βάσεων DNA με μήκος 50 έως 300 βάσεων συνήθως. Παράγονται από ένα δείγμα DNA κατά την αλληλούχιση με Next Generation Sequencing technologies.
Smith-Waterman (SmW)	Αλγόριθμος ευθυγράμμισης ζευγαριών αλληλουχιών βάσεων που βασίζεται στον Δυναμικό Προγραμματισμό.

Support Vector Machines (SVM)	Μηχανές Διανυσμάτων Υποστήριξης για ταξινόμηση δεδομένων σε κλάσεις.

Publications

Journals

1. Koliogeorgi, K., Xydis, S., Gaydadjiev, G., & Soudris, D. J. (2022). Dataflow Acceleration for Short Read Alignment on NGS data. IEEE Transactions on Computers.

2. Iliakis, K., Koliogeorgi, K., Litke, A., Varvarigou, T., & Soudris, D. (2022). GPU accelerated blockchain over key-value database transactions. IET Blockchain, 2(1), 1-12.

3. Leon, V., Mouselinos, S., Koliogeorgi, K., et al. (2020) “A TensorFlow Extension Framework for Optimized Generation of Hardware CNN Inference Engines” Technologies, 8(1), 6.

4. Zervakis, G., Koliogeorgi, K., et al. (2019). VADER: voltage-driven netlist pruning for cross-layer approximate arithmetic circuits. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 27(6), 1460-1464.

5. Tsoutsouras, V., Koliogeorgi, K., et al. (2017). An exploration framework for efficient high-level synthesis of support vector machines: case study on ECG arrhythmia detection for Xilinx Zynq SoC. Journal of Signal Processing Systems, 88(2), 127-147.

Conferences

1. Koliogeorgi, K., Soudris, D., Xydis, S.. Profile-Driven Banded Smith-Waterman acceleration for Short Read Alignment. Accepted in Design Automation Conference 2023.

2. Koliogeorgi, K., Mylonakis, D., Xydis, S., & Soudris, D. (2022, October). High Level Synthesis Acceleration of Change Detection in Multi-Temporal High Resolution Sentinel-2 Satellite Images. In 2022 IFIP/IEEE 30^th International Conference

on Very Large Scale Integration (VLSI-SoC) (pp. 1-6). IEEE.

3. Koliogeorgi, K., Keddous, F. E., Masouros, D., Chazapis, A., Aubrun, M., Xydis, S. & Soudris, D. (2021, August). FPGA acceleration in EVOLVE's Converged Cloud-HPC Infrastructure. In 2021 31st International Conference on Field-Programmable Logic and Applications (FPL) (pp. 376-377). IEEE.

4. Koliogeorgi, K., Xydis, S., & Soudris, D. (2021, June). FPGA Acceleration of Short Read Alignment. In Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (pp. 1-2).

5. Oroutzoglou, I., Masouros, D., Koliogeorgi, K., Xydis, S., & Soudris, D. (2020, May). Exploration of GPU sharing policies under GEMM workloads. In Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems (pp. 66-69).

6. Koliogeorgi, K., et al. "Dataflow Acceleration of Smith-Waterman with Traceback for High Throughput Next Generation Sequencing." 2019 29th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2019

7. Koliogeorgi, Konstantina, et al. "Optimizing SVM Classifier Through Approximate and High Level Synthesis Techniques." 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST). IEEE, 2019.

8. Zompakis, N., Anagnostos, D., Koliogeorgi, K., Zervakis, G., & Siozios, K. (2019, May). A Design Flow Framework for Fully-Connected Neural Networks Rapid Prototyping. In Proceedings of the International Conference on Omni-Layer Intelligent Systems (pp. 44-49).

9. Masouros, D., Koliogeorgi, et al. (2019, March). Co-design Implications of Cost-effective On-demand Acceleration for Cloud Healthcare Analytics: The AEGLE approach. In 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp. 622-625). IEEE.

10. Konstantina Koliogeorgi, Dimosthenis Masouros, Georgios Zervakis, Sotirios Xydis, Tobias Becker, Georgi Gaydadjiev, Dimitrios Soudris: AEGLE's Cloud Infrastructure for Resource Monitoring and Containerized Accelerated Analytics. ISVLSI 2017: 362-367

Workshops

1. Tsoutsouras, V., Koliogeorgi, K., Xydis, S., & Soudris, D. (2016). HLS code transformation strategies and directives exploration for FPGA accelerated ECG analysis. In Workshop in Reconfigurable Computing, Prague.

Bibliography

[1] S. Dash, S. K. Shakyawar, M. Sharma, and S. Kaushik, “Big data in healthcare: management, analysis and future prospects,” *Journal of Big Data*, vol. 6, no. 1, pp. 1–25, 2019.

[2] Xilinx, “Vivado design suite user guide: High-level synthesis, v2014.1.”

[3] M. Goodarzi, A. A. Fahimifar, and E. Shakeri Daryani, “New media and ideology: A critical perspective,” *Journal of Cyberspace Studies*, vol. 5, no. 2, pp. 121–140, 2021.

[4] X. Wang and Z. Yang, “Research on the youth group’s expectations for the future development of self-media while in the digital economy,” *Frontiers in Business, Economics and Management*, vol. 3, no. 3, pp. 43–48, 2022.

[5] J. Code, R. Ralph, and K. Forde, “A disorienting dilemma: teaching and learning in technology education during a time of crisis,” *Canadian Journal of Science, Mathematics and Technology Education*, vol. 22, no. 1, pp. 170–189, 2022.

[6] https://www.cancer.gov/publications/dictionaries/cancer-terms/def/electronic-medical record, “Electronic medical record by nih,”

[7] I. Subramanian, S. Verma, S. Kumar, A. Jere, and K. Anamika, “Multi-omics data integration, interpretation, and its application,” *Bioinformatics and biology insights*, vol. 14, p. 1177932219899051, 2020.

[8] A. R. Joyce and B. Ø. Palsson, “The model organism as a system: integrating’omics’ data sets,” *Nature reviews Molecular cell biology*, vol. 7, no. 3, pp. 198–210, 2006.

[9] J. K. Jansson and E. S. Baker, “A multi-omic future for microbiome studies,” *Nature microbiology*, vol. 1, no. 5, pp. 1–3, 2016.

[10] M. R. Bendre and V. R. Thool, “Analytics, challenges and applications in big data environment: a survey,” *Journal of Management Analytics*, vol. 3, no. 3, pp. 206–239, 2016.

[11] R. Ibrahim, M. Pasic, and G. M. Yousef, “Omics for personalized medicine: defining the current we swim in,” *Expert review of molecular diagnostics*, vol. 16, no. 7, pp. 719–722, 2016.

[12] K. Shameer, M. A. Badgeley, R. Miotto, B. S. Glicksberg, J. W. Morgan, and J. T. Dudley, “Translational bioinformatics in the era of real-time biomedical, health care and wellness data streams,” *Briefings in bioinformatics*, vol. 18, no. 1, pp. 105–124, 2017.

[13] Apache Software Foundation, “Hadoop.”

[14] W. Wu, W. Lin, C.-H. Hsu, and L. He, “Energy-efficient hadoop for big data analytics and computing: A systematic review and research insights,” *Future*

Generation Computer Systems, vol. 86, pp. 1351–1367, 2018.

[15] S. Allam, "Usage of hadoop and microsoft cloud in big data analytics: An exploratory study," *Sudhir Allam.(2018). USAGE OF HADOOP AND MICROSOFT CLOUD IN BIG DATA ANALYTICS: AN EXPLORATORY STUDY. International Journal of Innovations in Engineering Research and Technology*, vol. 5, no. 10, pp. 27–32, 2018.

[16] N. Dasgupta, *Practical big data analytics: Hands-on techniques to implement enterprise analytics and machine learning using Hadoop, Spark, NoSQL and R.*
Packt Publishing Ltd, 2018.

[17] B. Chambers and M. Zaharia, *Spark: The definitive guide: Big data processing made simple.*
" O'Reilly Media, Inc.", 2018.

[18] S. Alotaibi, R. Mehmood, I. Katib, O. Rana, and A. Albeshri, "Sehaa: A big data analytics tool for healthcare symptoms and diseases detection using twitter, apache spark, and machine learning," *Applied Sciences*, vol. 10, no. 4, p. 1398, 2020.

[19] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[20] N. Verma, D. Malhotra, and J. Singh, "Big data analytics for retail industry using mapreduce-apriori framework," *Journal of Management Analytics*, vol. 7, no. 3, pp. 424–442, 2020.

[21] Z. Lu, N. Wang, J. Wu, and M. Qiu, "Iotdem: An iot big data-oriented mapreduce performance prediction extended model in multiple edge clouds," *Journal of Parallel and Distributed Computing*, vol. 118, pp. 316–327, 2018.

[22] S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, and F. Herrera, "Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce," *Information Fusion*, vol. 42, pp. 51–61, 2018.

[23] S. P. Ahuja, S. Mani, and J. Zambrano, "A survey of the state of cloud computing in healthcare," *Network and Communication Technologies*, vol. 1, no. 2, p. 12, 2012.

[24] N. Sultan, "Making use of cloud computing for healthcare provision: Opportunities and challenges," *International Journal of Information Management*, vol. 34, no. 2, pp. 177–184, 2014.

[25] P. K. Sahoo, S. K. Mohapatra, and S.-L. Wu, "Sla based healthcare big data analysis and computing in cloud network," *Journal of Parallel and Distributed Computing*, vol. 119, pp. 121–135, 2018.

[26] https://www.illumina.com/products/by-type/informatics-products/dragen-bio-it platform.html, "Illumina dragen bio-it platform,"

[27] H. Asaadi, D. Khaldi, and B. Chapman, "A comparative survey of the hpc and big data paradigms: Analysis and experiments," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 423–432, IEEE, 2016.

[28] M. Anderson, S. Smith, N. Sundaram, M. Capotă, Z. Zhao, S. Dulloor, N. Satish, and T. L. Willke, "Bridging the gap between hpc and big data frameworks," *Proceedings of the VLDB Endowment*, vol. 10, no. 8, pp. 901–912, 2017.

[29] S. Usman, R. Mehmood, and I. Katib, “Big data and hpc convergence: The cutting edge and outlook,” in *International Conference on Smart Cities, Infrastructure, Technologies and Applications*, pp. 11–26, Springer, 2017.

[30] D. Elia, S. Fiore, and G. Aloisio, “Towards hpc and big data analytics convergence: Design and experimental evaluation of a hpda framework for escience at scale,” *IEEE Access*, vol. 9, pp. 73307–73326, 2021.

[31] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, *et al.*, “Open mpi: Goals, concept, and design of a next generation mpi implementation,” in *European Parallel Virtual Machine/Message Passing Interface Users’ Group Meeting*, pp. 97–104, Springer, 2004.

[32] L. Dagum and R. Menon, “Openmp: an industry standard api for shared-memory programming,” *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[33] E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto-Matias, “Swifold: Smith-waterman implementation on fpga with opencl for long dna sequences,” *BMC systems biology*, vol. 12, no. 5, pp. 43–53, 2018.

[34] Y. Liu, A. Wirawan, and B. Schmidt, “Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions,” *BMC bioinformatics*, vol. 14, no. 1, pp. 1–10, 2013.

[35] Y. Liu, T.-T. Tran, F. Lauenroth, and B. Schmidt, “Swaphi-ls: Smith-waterman algorithm on xeon phi coprocessors for long dna sequences,” in *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 257–265, IEEE, 2014.

[36] Y. Turakhia, G. Bejerano, and W. J. Dally, “Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly,” in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 199–213, ACM, 2018.

[37] NVIDIA, “Nvidia clara parabricks: Accelerate genome sequencing analysis,” *https://www.nvidia.com/en-us/clara/genomics/*.

[38] K.-H. Yu, A. L. Beam, and I. S. Kohane, “Artificial intelligence in healthcare,” *Nature biomedical engineering*, vol. 2, no. 10, pp. 719–731, 2018.

[39] G. Rong, A. Mendez, E. B. Assi, B. Zhao, and M. Sawan, “Artificial intelligence in healthcare: review and prediction case studies,” *Engineering*, vol. 6, no. 3, pp. 291–301, 2020.

[40] K. Shailaja, B. Seetharamulu, and M. Jabbar, “Machine learning in healthcare: A review,” in *2018 Second international conference on electronics, communication and aerospace technology (ICECA)*, pp. 910–914, IEEE, 2018.

[41] A. Qayyum, J. Qadir, M. Bilal, and A. Al-Fuqaha, “Secure and robust machine learning for healthcare: A survey,” *IEEE Reviews in Biomedical Engineering*, vol. 14, pp. 156–180, 2020.

[42] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, “Big data analytics: a survey,” *Journal of Big data*, vol. 2, no. 1, pp. 1–32, 2015.

[43] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, “Machine learning on big data:

Opportunities and challenges," *Neurocomputing*, vol. 237, pp. 350–361, 2017.

[44] A. B. Abacha and P. Zweigenbaum, "Means: A medical question-answering system combining nlp techniques and semantic web technologies," *Information processing & management*, vol. 51, no. 5, pp. 570–594, 2015.

[45] K. E. Henry, D. N. Hager, P. J. Pronovost, and S. Saria, "A targeted real-time early warning score (trewscore) for septic shock," *Science translational medicine*, vol. 7, no. 299, pp. 299ra122–299ra122, 2015.

[46] A. Ed-Daoudy and K. Maalmi, "Real-time machine learning for early detection of heart disease using big data approach," in *2019 international conference on wireless technologies, embedded and intelligent systems (WITS)*, pp. 1–5, IEEE, 2019.

[47] J. Stoitsis, I. Valavanis, S. G. Mougiakakou, S. Golemati, A. Nikita, and K. S. Nikita, "Computer aided diagnosis based on medical image processing and artificial intelligence methods," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 569, no. 2, pp. 591–595, 2006.

[48] I. Kiral-Kornek, S. Roy, E. Nurse, B. Mashford, P. Karoly, T. Carroll, D. Payne, S. Saha, S. Baldassano, T. O'Brien, *et al.*, "Epileptic seizure prediction using big data and deep learning: toward a mobile system," *EBioMedicine*, vol. 27, pp. 103–111, 2018.

[49] A. Rodríguez-Ruiz, E. Krupinski, J.-J. Mordang, K. Schilling, S. H. Heywang-Köbrunner, I. Sechopoulos, and R. M. Mann, "Detection of breast cancer with mammography: effect of an artificial intelligence support system," *Radiology*, vol. 290, no. 2, pp. 305–314, 2019.

[50] I. Sechopoulos, J. Teuwen, and R. Mann, "Artificial intelligence for breast cancer detection in mammography and digital breast tomosynthesis: State of the art," in *Seminars in Cancer Biology*, vol. 72, pp. 214–225, Elsevier, 2021.

[51] S. Safdar, S. Zafar, N. Zafar, and N. F. Khan, "Machine learning based decision support systems (dss) for heart disease diagnosis: a review," *Artificial Intelligence Review*, vol. 50, no. 4, pp. 597–623, 2018.

[52] K. Sakai and K. Yamada, "Machine learning studies on major brain diseases: 5-year trends of 2014–2018," *Japanese journal of radiology*, vol. 37, no. 1, pp. 34–72, 2019.

[53] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the hadoop ecosystem," *Journal of Big Data*, vol. 2, no. 1, pp. 1–36, 2015.

[54] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[55] D. Gillick, A. Faria, and J. DeNero, "Mapreduce: Distributed computing for machine learning," *Berkley, Dec*, vol. 18, 2006.

[56] K. Yue, H. Wu, X. Fu, J. Xu, Z. Yin, and W. Liu, "A data-intensive approach for discovering user similarities in social behavioral interactions based on the bayesian network," *Neurocomputing*, vol. 219, pp. 364–375, 2017.

[57] J. Wang, Y. Tang, M. Nguyen, and I. Altintas, “A scalable data science workflow approach for big data bayesian network learning,” in *2014 IEEE/ACM International Symposium on Big Data Computing*, pp. 16–25, IEEE, 2014.

[58] C.-T. Chu, S. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Ng, “Map-reduce for machine learning on multicore,” *Advances in neural information processing systems*, vol. 19, 2006.

[59] D. Luo, C. Ding, and H. Huang, “Parallelization with multiplicative algorithms for big data mining,” in *2012 IEEE 12th International Conference on Data Mining*, pp. 489–498, IEEE, 2012.

[60] I. Triguero, D. Peralta, J. Bacardit, S. García, and F. Herrera, “Mrpr: a mapreduce solution for prototype reduction in big data classification,” *neurocomputing*, vol. 150, pp. 331–345, 2015.

[61] A. Abdelaziz, M. Elhoseny, A. S. Salama, and A. Riad, “A machine learning model for improving healthcare services on cloud computing environment,” *Measurement*, vol. 119, pp. 117–128, 2018.

[62] F. Desai, D. Chowdhury, R. Kaur, M. Peeters, R. C. Arya, G. S. Wander, S. S. Gill, and R. Buyya, “Healthcloud: A system for monitoring health status of heart patients using machine learning and cloud computing,” *Internet of Things*, vol. 17, p. 100485, 2022.

[63] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, “Hardware for machine learning: Challenges and opportunities,” in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–8, IEEE, 2017.

[64] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, “Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic,” in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, IEEE, 2016.

[65] L. Du and Y. Du, “Hardware accelerator design for machine learning,” *Machine Learning—Advanced Techniques and Emerging Applications*, 2018.

[66] M. Shepovalov and V. Akella, “Fpga and gpu-based acceleration of ml workloads on amazon cloud-a case study using gradient boosted decision tree library,” *Integration*, vol. 70, pp. 1–9, 2020.

[67] K. Nagaraj, G. Sharvani, and A. Sridhar, “Emerging trend of big data analytics in bioinformatics: a literature review,” *International Journal of Bioinformatics Research and Applications*, vol. 14, no. 1-2, pp. 144–205, 2018.

[68] C. S. Greene, J. Tan, M. Ung, J. H. Moore, and C. Cheng, “Big data bioinformatics,” *Journal of cellular physiology*, vol. 229, no. 12, pp. 1896–1900, 2014.

[69] K. Y. He, D. Ge, and M. M. He, “Big data analytics for genomic medicine,” *International journal of molecular sciences*, vol. 18, no. 2, p. 412, 2017.

[70] M. A. Hamburg and F. S. Collins, “The path to personalized medicine,” *New England Journal of Medicine*, vol. 363, no. 4, pp. 301–304, 2010.

[71] P. C. Ng and E. F. Kirkness, “Whole genome sequencing,” *Genetic variation*, pp. 215–226, 2010.

[72] G. C. Kennedy, H. Matsuzaki, S. Dong, W.-m. Liu, J. Huang, G. Liu, X. Su, M. Cao, W. Chen, J. Zhang, *et al.*, “Large-scale genotyping of complex dna,”

Nature biotechnology, vol. 21, no. 10, pp. 1233–1237, 2003.

[73] L. M. Starita, N. Ahituv, M. J. Dunham, J. O. Kitzman, F. P. Roth, G. Seelig, J. Shendure, and D. M. Fowler, "Variant interpretation: functional assays to the rescue," *The American Journal of Human Genetics*, vol. 101, no. 3, pp. 315–325, 2017.

[74] T. Lappalainen, A. J. Scott, M. Brandt, and I. M. Hall, "Genomic analysis in the age of human genome sequencing," *Cell*, vol. 177, no. 1, pp. 70–84, 2019.

[75] V. N. Kristensen, O. C. Lingjærde, H. G. Russnes, H. K. M. Vollan, A. Frigessi, and A.-L. Børresen-Dale, "Principles and methods of integrative genomic analyses in cancer," *Nature Reviews Cancer*, vol. 14, no. 5, pp. 299–313, 2014.

[76] A. O'Driscoll, J. Daugelaite, and R. D. Sleator, "'big data', hadoop and cloud computing in genomics," *Journal of biomedical informatics*, vol. 46, no. 5, pp. 774–781, 2013.

[77] R. Gullapalli, M. Lyons-Weiler, P. Petrosko, R. Dhir, M. Becich, and W. LaFramboise, "Clinical integration of next-generation sequencing technology," *Clinics in laboratory medicine*, vol. 32, no. 4, pp. 585–599, 2012.

[78] K. Yelick, A. Buluç, M. Awan, A. Azad, B. Brock, R. Egan, S. Ekanayake, M. Ellis, E. Georganas, G. Guidi, *et al.*, "The parallelism motifs of genomic data analysis," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190394, 2020.

[79] R. C. Taylor, "An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics," *BMC bioinformatics*, vol. 11, no. 12, pp. 1–6, 2010.

[80] Q. Zou, X.-B. Li, W.-R. Jiang, Z.-Y. Lin, G.-L. Li, and K. Chen, "Survey of mapreduce frame operation in bioinformatics," *Briefings in bioinformatics*, vol. 15, no. 4, pp. 637–647, 2014.

[81] Q. B. Baker, W. Al-Rashdan, and Y. Jararweh, "Cloud-based tools for next-generation sequencing data analysis," in *2018 fifth international conference on social networks analysis, management and security (SNAMS)*, pp. 99–105, IEEE, 2018.

[82] B. Calabrese and M. Cannataro, "Cloud computing in bioinformatics: current solutions and challenges," 2016.

[83] E. Afgan, C. Sloggett, N. Goonasekera, I. Makunin, D. Benson, M. Crowe, S. Gladman, Y. Kowsar, M. Pheasant, R. Horst, *et al.*, "Genomics virtual laboratory: a practical bioinformatics workbench for the cloud," *PloS one*, vol. 10, no. 10, p. e0140829, 2015.

[84] B. Liu, R. K. Madduri, B. Sotomayor, K. Chard, L. Lacinski, U. J. Dave, J. Li, C. Liu, and I. T. Foster, "Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses," *Journal of biomedical informatics*, vol. 49, pp. 119–133, 2014.

[85] L. Dai, X. Gao, Y. Guo, J. Xiao, and Z. Zhang, "Bioinformatics clouds for big data manipulation," *Biology direct*, vol. 7, no. 1, pp. 1–7, 2012.

[86] T. J. Ham, D. Bruns-Smith, B. Sweeney, Y. Lee, S. H. Seo, U. G. Song, Y. H. Oh, K. Asanovic, J. W. Lee, and L. W. Wills, "Genesis: a hardware acceleration

framework for genomic data analysis," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 254–267, IEEE, 2020.

[87] B. Langmead and A. Nellore, "Cloud computing for genomic data analysis and collaboration," *Nature Reviews Genetics*, vol. 19, no. 4, pp. 208–219, 2018.

[88] E. Fernandez, W. Najjar, and S. Lonardi, "String matching in hardware using the fm-index," in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 218–225, IEEE, 2011.

[89] R. Langarita, A. Armejach, J. Setoain, P. Ibanez-Marin, J. Alastruey-Benedé, and M. Moretó, "Compressed sparse fm-index: Fast sequence alignment using large k-steps," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 19, no. 1, pp. 355–368, 2020.

[90] M. Alser, H. Hassan, A. Kumar, O. Mutlu, and C. Alkan, "Shouji: a fast and efficient pre-alignment filter for sequence alignment," *Bioinformatics*, vol. 35, no. 21, pp. 4255–4263, 2019.

[91] F. Hach, I. Sarrafi, F. Hormozdiari, C. Alkan, E. E. Eichler, and S. C. Sahinalp, "mrsfast-ultra: a compact, snp-aware mapper for high performance sequencing applications," *Nucleic acids research*, vol. 42, no. W1, pp. W494–W500, 2014.

[92] B. Liu, D. Guan, M. Teng, and Y. Wang, "rhat: fast alignment of noisy long reads with regional hashing," *Bioinformatics*, vol. 32, no. 11, pp. 1625–1631, 2016.

[93] M. Alser, Z. Bingöl, D. S. Cali, J. Kim, S. Ghose, C. Alkan, and O. Mutlu, "Accelerating genome analysis: A primer on an ongoing journey," *IEEE Micro*, vol. 40, no. 5, pp. 65–75, 2020.

[94] W. Huangfu, S. Li, X. Hu, and Y. Xie, "Radar: a 3d-reram based dna alignment accelerator architecture," in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

[95] J. S. Kim, D. Senol Cali, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "Grim-filter: Fast seed location filtering in dna read mapping using processing-in-memory technologies," *BMC genomics*, vol. 19, no. 2, pp. 23–40, 2018.

[96] S. Gupta, M. Imani, B. Khaleghi, V. Kumar, and T. Rosing, "Rapid: A reram processing in-memory architecture for dna sequence alignment," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, IEEE, 2019.

[97] A. Cilardo and L. Gallo, "Improving multibank memory access parallelism with lattice-based partitioning," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 4, p. 45, 2015.

[98] Y. Ben-Asher and N. Rotem, "Automatic memory partitioning: increasing memory parallelism via data structure partitioning," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 155–162, ACM, 2010.

[99] J. Cong, P. Li, B. Xiao, and P. Zhang, "An optimal microarchitecture for stencil computation acceleration based on non-uniform partitioning of data reuse buffers," in *Proceedings of the 51st Annual Design Automation Conference*,

pp. 1–6, ACM, 2014.

[100] J. Cong, W. Jiang, B. Liu, and Y. Zou, "Automatic memory partitioning and scheduling for throughput and power optimization," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 16, no. 2, p. 15, 2011.

[101] Y. Wang, P. Zhang, X. Cheng, and J. Cong, "An integrated and automated memory optimization flow for fpga behavioral synthesis," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pp. 257–262, IEEE, 2012.

[102] J. Su, F. Yang, X. Zeng, and D. Zhou, "Efficient memory partitioning for parallel data access via data reuse," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 138–147, ACM, 2016.

[103] S. Xydis, G. Palermo, V. Zaccaria, and C. Silvano, "SPIRIT: spectral-aware pareto iterative refinement optimization for supervised high-level synthesis," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 34, no. 1, pp. 155–159, 2015.

[104] H. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013*, pp. 50:1–50:7, 2013.

[105] B. C. Schäfer and K. Wakabayashi, "Divide and conquer high-level synthesis design space exploration," *ACM Trans. Design Autom. Electr. Syst.*, vol. 17, no. 3, p. 29, 2012.

[106] S. Xydis, K. Z. Pekmestzi, D. Soudris, and G. Economakos, "Compiler-in-the-loop exploration during datapath synthesis for higher quality delay-area trade-offs," *ACM Trans. Design Autom. Electr. Syst.*, vol. 18, no. 1, p. 11, 2012.

[107] A. A. Nacci, V. Rana, F. Bruschi, D. Sciuto, I. Beretta, and D. Atienza, "A high-level synthesis flow for the implementation of iterative stencil loop algorithms on fpga devices," in *Proceedings of the 50th Annual Design Automation Conference*, p. 52, ACM, 2013.

[108] P. Li, Y. Wang, P. Zhang, G. Luo, T. Wang, and J. Cong, "Memory partitioning and scheduling co-optimization in behavioral synthesis," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 488–495, ACM, 2012.

[109] G. Zhong, V. Venkataramani, Y. Liang, T. Mitra, and S. Niar, "Design space exploration of multiple loops on fpgas using high level synthesis," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pp. 456–463, IEEE, 2014.

[110] J. Cong, P. Zhang, and Y. Zou, "Optimizing memory hierarchy allocation with loop transformations for high-level synthesis," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 1233–1238, ACM, 2012.

[111] A. Cilardo and L. Gallo, "Interplay of loop unrolling and multidimensional memory partitioning in hls," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 163–168, IEEE, 2015.

[112] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.

[113] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2,"

Nature methods, vol. 9, no. 4, p. 357, 2012.

[114] M. Šošić and M. Šikić, “Edlib: a c/c++ library for fast, exact sequence alignment using edit distance,” *Bioinformatics*, vol. 33, no. 9, pp. 1394–1395, 2017.

[115] S. Marco-Sola, J. C. Moure, M. Moreto, and A. Espinosa, “Fast gap-affine pairwise alignment using the wavefront algorithm,” *Bioinformatics*, vol. 37, no. 4, pp. 456–463, 2021.

[116] H. Suzuki and M. Kasahara, “Introducing difference recurrence relations for faster semi-global alignment of long sequences,” *BMC bioinformatics*, vol. 19, no. 1, pp. 33–47, 2018.

[117] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[118] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[119] M. Alser, T. Shahroodi, J. Gómez-Luna, C. Alkan, and O. Mutlu, “Sneakysnake: a fast and accurate universal genome pre-alignment filter for cpus, gpus and fpgas,” *Bioinformatics*, vol. 36, no. 22-23, pp. 5282–5290, 2020.

[120] W. R. Pearson, “Searching protein sequence libraries: comparison of the sensitivity and selectivity of the smith-waterman and fasta algorithms,” *Genomics*, vol. 11, no. 3, pp. 635–650, 1991.

[121] H.-C. Ng, S. Liu, and W. Luk, “Reconfigurable acceleration of genetic sequence alignment: A survey of two decades of efforts,” in *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, pp. 1–8, IEEE, 2017.

[122] K. Puttegowda, W. Worek, N. Pappas, A. Dandapani, P. Athanas, and A. Dickerman, “A run-time reconfigurable system for gene-sequence searching,” in *VLSI Design, 2003. Proceedings. 16th International Conference on*, pp. 561–566, IEEE, 2003.

[123] M. Gok and C. Yilmaz, “Efficient cell designs for systolic smith-waterman implementations,” in *Field Programmable Logic and Applications, 2006. FPL’06. International Conference on*, pp. 1–4, IEEE, 2006.

[124] P. Zhang, G. Tan, and G. R. Gao, “Implementation of the smith-waterman algorithm on a reconfigurable supercomputing platform,” in *Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications: held in conjunction with SC07*, pp. 39–48, ACM, 2007.

[125] S. A. Guccione and E. Keller, “Gene matching using jbits,” in *International Conference on Field Programmable Logic and Applications*, pp. 1168–1171, Springer, 2002.

[126] P. Faes, B. Minnaert, M. Christiaens, E. Bonnet, Y. Saeys, D. Stroobandt, and Y. Van de Peer, “Scalable hardware accelerator for comparing dna and protein sequences,” in *Proceedings of the 1st international conference on Scalable information systems*, pp. 33–es, 2006.

[127] X. Jiang, X. Liu, L. Xu, P. Zhang, and N. Sun, “A reconfigurable accelerator for

smith–waterman algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 12, pp. 1077–1081, 2007.

[128] K. Benkrid, Y. Liu, and A. Benkrid, "A highly parameterized and efficient fpga-based skeleton for pairwise biological sequence alignment," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 4, pp. 561–570, 2009.

[129] Z. Nawaz, M. Nadeem, H. van Someren, and K. Bertels, "A parallel fpga design of the smith-waterman traceback," in *2010 International Conference on Field-Programmable Technology*, pp. 454–459, IEEE, 2010.

[130] S. Lloyd and Q. O. Snell, "Hardware accelerated sequence alignment with traceback," *International Journal of Reconfigurable Computing*, vol. 2009, p. 9, 2009.

[131] Y.-T. Chen, J. Cong, J. Lei, and P. Wei, "A novel high-throughput acceleration engine for read alignment," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 199–202, IEEE, 2015.

[132] C. B. Olson, M. Kim, C. Clauson, B. Kogon, C. Ebeling, S. Hauck, and W. L. Ruzzo, "Hardware acceleration of short read mapping," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pp. 161–168, IEEE, 2012.

[133] N. Homer, B. Merriman, and S. F. Nelson, "Bfast: an alignment tool for large scale genome resequencing," *PloS one*, vol. 4, no. 11, p. e7767, 2009.

[134] D. Fujiki, A. Subramaniyan, T. Zhang, Y. Zeng, R. Das, D. Blaauw, and S. Narayanasamy, "Genax: A genome sequencing accelerator," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 69–82, IEEE, 2018.

[135] D. Fujiki, S. Wu, N. Ozog, K. Goliya, D. Blaauw, S. Narayanasamy, and R. Das, "Seedex: A genome sequencing accelerator for optimal alignments in subminimal space," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 937–950, IEEE, 2020.

[136] D. S. Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand, *et al.*, "Genasm: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 951–966, IEEE, 2020.

[137] S. S. Banerjee, M. El-Hadedy, J. B. Lim, Z. T. Kalbarczyk, D. Chen, S. S. Lumetta, and R. K. Iyer, "Asap: Accelerated short-read alignment on programmable hardware," *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 331–346, 2018.

[138] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2015.

[139] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4,

pp. 473–491, 2011.

[140] K. Koliogeorgi, “Optimizing ecg signal analysis by building fpga-based accelerators using high level synthesis,” 2016.

[141] S. Soldavini, S. L. Alarcón, and M. Łukowiak, “Using reduced graphs for efficient hls scheduling,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2020.

[142] J. Jung and T. Kim, “Scheduling and resource binding algorithm considering timing variation,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 2, pp. 205–216, 2009.

[143] P. Kollig and B. Al-Hashimi, “Simultaneous scheduling, allocation and binding in high level synthesis,” *Electronics Letters*, vol. 33, no. 18, pp. 1516–1518, 1997.

[144] J. E. Stone, D. Gohara, and G. Shi, “Opencl: A parallel programming standard for heterogeneous computing systems,” *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.

[145] T. Feist, “Vivado design suite,” *White Paper*, vol. 5, 2012.

[146] V. Kathail, “Xilinx vitis unified software platform,” in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 173–174, 2020.

[147] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, “Legup: high-level synthesis for fpga-based processor/accelerator systems,” in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 33–36, 2011.

[148] T. J. Todman, G. A. Constantinides, S. J. Wilton, O. Mencer, W. Luk, and P. Y. Cheung, “Reconfigurable computing: architectures and design methods,” *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2005.

[149] O. Pell and O. Mencer, “Surviving the end of frequency scaling with reconfigurable dataflow computing,” *ACM SIGARCH Computer Architecture News*, vol. 39, no. 4, pp. 60–65, 2011.

[150] J. D. Watson, “The human genome project: past, present, and future,” *Science*, vol. 248, no. 4951, pp. 44–49, 1990.

[151] M. L. Metzker, “Sequencing technologies—the next generation,” *Nature reviews genetics*, vol. 11, no. 1, pp. 31–46, 2010.

[152] N. M. Luscombe, D. Greenbaum, and M. Gerstein, “What is bioinformatics? a proposed definition and overview of the field,” *Methods of information in medicine*, vol. 40, no. 04, pp. 346–358, 2001.

[153] S. Goodwin, J. D. McPherson, and W. R. McCombie, “Coming of age: ten years of next-generation sequencing technologies,” *Nature Reviews Genetics*, vol. 17, no. 6, pp. 333–351, 2016.

[154] M. Kchouk, J.-F. Gibrat, and M. Elloumi, “Generations of sequencing technologies: from first to next generation,” *Biology and Medicine*, vol. 9, no. 3, 2017.

[155] F. Sanger, S. Nicklen, and A. R. Coulson, “Dna sequencing with chain-terminating inhibitors,” *Proceedings of the national academy of sciences*, vol. 74, no. 12, pp. 5463–5467, 1977.

[156] J. K. Kulski, "Next-generation sequencing—an overview of the history, tools, and "omic" applications," *Next Generation Sequencing–Advances, Applications and Challenges*, pp. 3–60, 2016.

[157] J. Shendure and H. Ji, "Next-generation dna sequencing," *Nature biotechnology*, vol. 26, no. 10, pp. 1135–1145, 2008.

[158] S. Balasubramanian, "Solexa sequencing: Decoding genomes on a population scale," *Clinical chemistry*, vol. 61, no. 1, pp. 21–24, 2015.

[159] A. Rhoads and K. F. Au, "Pacbio sequencing and its applications," *Genomics, proteomics & bioinformatics*, vol. 13, no. 5, pp. 278–289, 2015.

[160] A. S. Mikheyev and M. M. Tin, "A first look at the oxford nanopore minion sequencer," *Molecular ecology resources*, vol. 14, no. 6, pp. 1097–1102, 2014.

[161] C. Alkan, B. P. Coe, and E. E. Eichler, "Genome structural variation discovery and genotyping," *Nature reviews genetics*, vol. 12, no. 5, pp. 363–376, 2011.

[162] H. Li, "Toward better understanding of artifacts in variant calling from high-coverage samples," *Bioinformatics*, vol. 30, no. 20, pp. 2843–2851, 2014.

[163] J. P. Carulli, M. Artinger, P. M. Swain, C. D. Root, L. Chee, C. Tulig, J. Guerin, M. Osborne, G. Stein, J. Lian, *et al.*, "High throughput analysis of differential gene expression," *Journal of Cellular Biochemistry*, vol. 72, no. S30–31, pp. 286–296, 1998.

[164] S. J. Gould, *Ontogeny and phylogeny.*
Harvard University Press, 1977.

[165] O. Morozova, M. Hirst, and M. A. Marra, "Applications of new sequencing technologies for transcriptome analysis," *Annual review of genomics and human genetics*, vol. 10, pp. 135–151, 2009.

[166] A. Bird, "Perceptions of epigenetics," *Nature*, vol. 447, no. 7143, p. 396, 2007.

[167] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants," *Nucleic acids research*, vol. 38, no. 6, pp. 1767–1771, 2010.

[168] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, "The sequence alignment/map format and samtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[169] O. U. Sezerman, E. Ulgen, N. Seymen, and I. M. Durasi, "Bioinformatics workflows for genomic variant discovery, interpretation and prioritization," in *Bioinformatics Tools for Detection and Clinical Interpretation of Genomic Variations*, IntechOpen, 2019.

[170] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[171] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short dna sequences to the human genome," *Genome biology*, vol. 10, no. 3, p. R25, 2009.

[172] R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang, "Soap2: an improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, pp. 1966–1967, 2009.

[173] M. Alser, J. Rotman, D. Deshpande, K. Taraszka, H. Shi, P. I. Baykal, H. T. Yang,

V. Xue, S. Knyazev, B. D. Singer, *et al.*, "Technology dictates algorithms: recent developments in read alignment," *Genome biology*, vol. 22, no. 1, pp. 1–34, 2021.

[174] G. Manzini, "An analysis of the burrows—wheeler transform," *Journal of the ACM (JACM)*, vol. 48, no. 3, pp. 407–430, 2001.

[175] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[176] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM journal of research and development*, vol. 31, no. 2, pp. 249–260, 1987.

[177] V. Likic, "The needleman-wunsch algorithm for sequence alignment," *Lecture given at the 7th Melbourne Bioinformatics Course, Bi021 Molecular Science and Biotechnology Institute, University of Melbourne*, pp. 1–46, 2008.

[178] V. Tsoutsouras, K. Koliogeorgi, S. Xydis, and D. Soudris, "An exploration framework for efficient high-level synthesis of support vector machines: Case study on ecg arrhythmia detection for xilinx zynq soc," *Journal of Signal Processing Systems*, vol. 88, no. 2, pp. 127–147, 2017.

[179] K. Koliogeorgi, G. Zervakis, D. Anagnostos, N. Zompakis, and K. Siozios, "Optimizing svm classifier through approximate and high level synthesis techniques," in *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–4, IEEE, 2019.

[180] V. Tsoutsouras, "Design methodologies for resource management of many-core embedded systems," 2018.

[181] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[182] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of machine learning research*, vol. 2, no. Nov, pp. 45–66, 2001.

[183] C. H. Wan, L. H. Lee, R. Rajkumar, and D. Isa, "A hybrid text classification approach with low dependency on parameter by integrating k-nearest neighbor and support vector machine," *Expert Systems with Applications*, vol. 39, no. 15, pp. 11880–11888, 2012.

[184] T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler, "Support vector machine classification and validation of cancer tissue samples using microarray expression data," *Bioinformatics*, vol. 16, no. 10, pp. 906–914, 2000.

[185] G. Orrù, W. Pettersson-Yeo, A. F. Marquand, G. Sartori, and A. Mechelli, "Using support vector machine to identify imaging biomarkers of neurological and psychiatric disease: a critical review," *Neuroscience & Biobehavioral Reviews*, vol. 36, no. 4, pp. 1140–1152, 2012.

[186] S. Tong and E. Chang, "Support vector machine active learning for image retrieval," in *Proceedings of the ninth ACM international conference on Multimedia*, pp. 107–118, ACM, 2001.

[187] M. Volpi, D. Tuia, F. Bovolo, M. Kanevski, and L. Bruzzone, "Supervised change detection in vhr images using contextual information and support vector machines," *International Journal of Applied Earth Observation and Geoinforma-*

tion, vol. 20, pp. 77–85, 2013.

[188] Y. Zhang, S. Wang, G. Ji, and Z. Dong, "An mr brain images classifier system via particle swarm optimization and kernel support vector machine," *The Scientific World Journal*, vol. 2013, 2013.

[189] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International Workshop on Ambient Assisted Living*, pp. 216–223, Springer, 2012.

[190] Y. Tang, "Deep learning using linear support vector machines," *arXiv preprint arXiv:1306.0239*, 2013.

[191] M. Papadonikolakis and C.-S. Bouganis, "A novel fpga-based svm classifier," in *Field-Programmable Technology (FPT), 2010 International Conference on*, pp. 283–286, IEEE, 2010.

[192] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. P. Graf, "A massively parallel fpga-based coprocessor for support vector machines," in *Field Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on*, pp. 115–122, IEEE, 2009.

[193] M. Shoaib, N. K. Jha, and N. Verma, "Algorithm-driven architectural design space exploration of domain-specific medical-sensor processors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 10, pp. 1849–1862, 2013.

[194] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Design Automation Conf.*, June 2015.

[195] H. A. F. Almurib, T. N. Kumar, and F. Lombardi, "Inexact designs for approximate low power addition by cell replacement," in *Design, Automation Test in Europe*, 2016.

[196] G. Zervakis, S. Xydis, K. Tsoumanis, D. Soudris, and K. Pekmestzi, "Hybrid approximate multiplier architectures for improved power-accuracy trade-offs," in *International Symposium on Low Power Electronics and Design*, pp. 79–84, July 2015.

[197] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, "Design of approximate radix-4 booth multipliers for error-tolerant computing," *IEEE Trans. Comput.*, vol. 66, pp. 1435–1441, Aug 2017.

[198] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, "Design-efficient approximate multiplication circuits through partial product perforation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 3105–3117, Oct 2016.

[199] A. Lingamneni, C. Enz, K. Palem, and C. Piguet, "Synthesizing parsimonious inexact circuits through probabilistic design techniques," *ACM Trans. Embed. Comput. Syst.*, vol. 12, pp. 93:1–93:26, May 2013.

[200] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high level synthesis for approximate computing," in *Design Automation Conference*, pp. 104:1–104:6, 2015.

[201] G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi, "Multi-level approximate

accelerator synthesis under voltage island constraints," *IEEE Trans. Circuits Syst. II*, pp. 1–1, 2018.

[202] R. M. Rangayyan and N. P. Reddy, "Biomedical signal analysis: a case-study approach," *Annals of Biomedical Engineering*, vol. 30, no. 7, pp. 983–983, 2002.

[203] E. D. Übeyli, "Ecg beats classification using multiclass support vector machines with error correcting output codes," *Digital Signal Processing*, vol. 17, no. 3, pp. 675–684, 2007.

[204] H. Zhang and L.-Q. Zhang, "Ecg analysis based on pca and support vector machines," in *Neural Networks and Brain, 2005. ICNN&B'05. International Conference on*, vol. 2, pp. 743–747, IEEE, 2005.

[205] G. B. Moody and R. G. Mark, "The impact of the mit-bih arrhythmia database," *Engineering in Medicine and Biology Magazine, IEEE*, vol. 20, no. 3, pp. 45–50, 2001.

[206] M. C. Ramon, "Intel galileo and intel galileo gen 2," in *Intel® Galileo and Intel® Galileo Gen 2*, pp. 1–33, Springer, 2014.

[207] D. Azariadi, V. Tsoutsouras, S. Xydis, and D. Soudris, "Ecg signal analysis and arrhythmia detection on iot wearable medical devices," in *Modern Circuits and Systems Technologies (MOCAST), 2016 5th International Conference on*, pp. 1–4, IEEE, 2016.

[208] F. Digilent's ZedBoard Zynq, "Dev. board documentation."

[209] X. Zynq, "7000 all programmable soc zc702 evaluation kit," 2015.

[210] S. Xydis, G. Economakos, D. Soudris, and K. Z. Pekmestzi, "High performance and area efficient flexible DSP datapath synthesis," *IEEE Trans. VLSI Syst.*, vol. 19, no. 3, pp. 429–442, 2011.

[211] S. Xydis, I. S. Triantafyllou, G. Economakos, and K. Z. Pekmestzi, "Flexible datapath synthesis through arithmetically optimized operation chaining," in *NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2009, San Francisco, California, USA, July 29 - August 1, 2009*, pp. 407–414, 2009.

[212] H. Parandeh-Afshar, A. K. Verma, P. Brisk, and P. Ienne, "Improving FPGA performance for carry-save arithmetic," *IEEE Trans. VLSI Syst.*, vol. 18, no. 4, pp. 578–590, 2010.

[213] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 2, p. 24, 2013.

[214] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng. (ESEC/FSE)*, 2011.

[215] G. Coley, "Beagleboard system reference manual," *BeagleBoard. org, December*, p. 81, 2009.

[216] Z. J. Xu, "Ls2085/8a freescale's new qorlq layerscape communications processor," in *Hot Chips 27 Symposium (HCS), 2015 IEEE*, pp. 1–25, IEEE, 2015.

[217] K. Koliogeorgi, N. Voss, S. Fytraki, S. Xydis, G. Gaydadjiev, and D. Soudris, "Dataflow acceleration of smith-waterman with traceback for high throughput

next generation sequencing," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 74–80, IEEE, 2019.

[218] K. Koliogeorgi, S. Xydis, G. Gaydadjiev, and D. J. Soudris, "Dataflow acceleration for short read alignment on ngs data," *IEEE Transactions on Computers*, 2022.

[219] S. T. Park and J. Kim, "Trends in next-generation sequencing and a new era for whole genome sequencing," *International neurourology journal*, vol. 20, no. Suppl 2, p. S76, 2016.

[220] S. J. Bielinski, J. E. Olson, J. Pathak, R. M. Weinshilboum, L. Wang, K. J. Lyke, E. Ryu, P. V. Targonski, M. D. Van Norstrand, M. A. Hathcock, *et al.*, "Preemptive genotyping for personalized medicine: design of the right drug, right dose, right time—using genomic data to individualize treatment protocol," in *Mayo Clinic Proceedings*, vol. 89, pp. 25–33, Elsevier, 2014.

[221] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. Del Angel, M. A. Rivas, M. Hanna, *et al.*, "A framework for variation discovery and genotyping using next-generation dna sequencing data," *Nature genetics*, vol. 43, no. 5, p. 491, 2011.

[222] M. D. Robinson, D. J. McCarthy, and G. K. Smyth, "edger: a bioconductor package for differential expression analysis of digital gene expression data," *Bioinformatics*, vol. 26, no. 1, pp. 139–140, 2010.

[223] B. Schmidt and A. Hildebrandt, "Next-generation sequencing: big data meets high performance computing," *Drug discovery today*, vol. 22, no. 4, pp. 712–717, 2017.

[224] http://www.novocraft.com/products/novoalign/

[225] Y. Liu and B. Schmidt, "Long read alignment based on maximal exact match seeds," *Bioinformatics*, vol. 28, no. 18, pp. i318–i324, 2012.

[226] H. Ye, J. Meehan, W. Tong, and H. Hong, "Alignment of short reads: a crucial step for application of next-generation sequencing data in precision medicine," *Pharmaceutics*, vol. 7, no. 4, pp. 523–541, 2015.

[227] N. Ahmed, K. Bertels, and Z. Al-Ars, "A comparison of seed-and-extend techniques in modern dna read alignment algorithms," in *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 1421–1428, IEEE, 2016.

[228] Y.-T. Chen, J. Cong, Z. Fang, J. Lei, and P. Wei, "When apache spark meets fpgas: a case study for next-generation dna sequencing acceleration," in *The 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[229] J. Arram, K. H. Tsoi, W. Luk, and P. Jiang, "Reconfigurable acceleration of short read mapping," in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 210–217, IEEE, 2013.

[230] J. Arram, K. H. Tsoi, W. Luk, and P. Jiang, "Hardware acceleration of genetic sequence alignment," in *International Symposium on Applied Reconfigurable Computing*, pp. 13–24, Springer, 2013.

[231] H.-C. Ng, S. Liu, I. Coleman, R. S. Chu, M.-C. Yue, and W. Luk, "Acceleration of short read alignment with runtime reconfiguration," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, pp. 256–262, 2020.

[232] H.-C. Ng, I. Coleman, S. Liu, and W. Luk, "Reconfigurable acceleration of short read mapping with biological consideration," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 229–239, 2021.

[233] W. Tang, W. Wang, B. Duan, C. Zhang, G. Tan, P. Zhang, and N. Sun, "Accelerating millions of short reads mapping on a heterogeneous architecture with fpga accelerator," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pp. 184–187, IEEE, 2012.

[234] E. J. Houtgast, V.-M. Sima, K. Bertels, and Z. Al-Ars, "An fpga-based systolic array to accelerate the bwa-mem genomic mapping algorithm," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on*, pp. 221–227, IEEE, 2015.

[235] https://github.com/maxeler/Smith-Waterman, "Smithwaterman demo on maxeler github,"

[236] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[237] http://bowtie-bio.sourceforge.net/bowtie2/manual.shtmlmain arguments, "Bowtie2 manual for alignment configuration,"

[238] T. C. Glenn, "Field guide to next-generation dna sequencers," *Molecular ecology resources*, vol. 11, no. 5, pp. 759–769, 2011.

[239] N. Trifunovic, V. Milutinovic, N. Korolija, and G. Gaydadjiev, "An appgallery for dataflow computing," *Journal of Big Data*, vol. 3, no. 1, pp. 1–30, 2016.

[240] Y. Guo, X. Ding, Y. Shen, G. J. Lyon, and K. Wang, "Seqmule: automated pipeline for analysis of human exome/genome sequencing data," *Scientific reports*, vol. 5, p. 14283, 2015.

[241] Z. D. Stephens, M. E. Hudson, L. S. Mainzer, M. Taschuk, M. R. Weber, and R. K. Iyer, "Simulating next-generation sequencing datasets from empirical mutation and sequencing models," *PloS one*, vol. 11, no. 11, p. e0167047, 2016.

[242] M. G. Ross, C. Russ, M. Costello, A. Hollinger, N. J. Lennon, R. Hegarty, C. Nusbaum, and D. B. Jaffe, "Characterizing and measuring bias in sequence data," *Genome biology*, vol. 14, no. 5, pp. 1–20, 2013.

[243] http://www.aegle uhealth.eu/en/, "Aegle: An analytics framework for integrated and personalized healthcare services in europe,"

[244] P. Baliakas, A. Hadzidimitriou, L. A. Sutton, D. Rossi, E. Minga, N. Villamor, M. Larrayoz, J. Kmínková, A. Agathangelidis, Z. Davis, *et al.*, "Recurrent mutations refine prognosis in chronic lymphocytic leukemia," *Leukemia*, vol. 29, no. 2, pp. 329–336, 2015.

[245] C. L. Richards, O. Bossdorf, and K. J. Verhoeven, "Understanding natural epigenetic variation," *The New Phytologist*, vol. 187, no. 3, pp. 562–564, 2010.

[246] K. Koliogeorgi, S. Xydis, G. Gaydadjiev, and D. Soudris, "Gandafl: Dataflow acceleration for short read alignment on ngs data," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 3018–3031, 2022.

[247] K.-M. Chao, W. R. Pearson, and W. Miller, "Aligning two sequences within a specified diagonal band," *Bioinformatics*, vol. 8, no. 5, pp. 481–487, 1992.

[248] Y.-L. Liao, Y.-C. Li, N.-C. Chen, and Y.-C. Lu, "Adaptively banded smith-

waterman algorithm for long reads and its hardware accelerator," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 1–9, IEEE, 2018.

[249] H. Suzuki and M. Kasahara, "Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming," *BioRxiv*, p. 130633, 2017.

[250] M. A. Eberle, E. Fritzilas, P. Krusche, M. Källberg, B. L. Moore, M. A. Bekritsky, Z. Iqbal, H.-Y. Chuang, S. J. Humphray, A. L. Halpern, *et al.*, "A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree," *Genome research*, vol. 27, no. 1, pp. 157–164, 2017.

[251] G. Eraslan, Ž. Avsec, J. Gagneur, and F. J. Theis, "Deep learning: new computational modelling techniques for genomics," *Nature Reviews Genetics*, vol. 20, no. 7, pp. 389–403, 2019.

[252] T. A. Liu, H. Zhu, H. Chen, J. F. Arevalo, F. K. Hui, H. Y. Paul, J. Wei, M. Unberath, and Z. M. Correa, "Gene expression profile prediction in uveal melanoma using deep learning: a pilot study for the development of an alternative survival prediction tool," *Ophthalmology Retina*, vol. 4, no. 12, pp. 1213–1215, 2020.

[253] J. Zhou and O. G. Troyanskaya, "Predicting effects of noncoding variants with deep learning–based sequence model," *Nature methods*, vol. 12, no. 10, pp. 931–934, 2015.

[254] V. Boža, B. Brejová, and T. Vinař, "Deepnano: deep recurrent neural networks for base calling in minion nanopore reads," *PloS one*, vol. 12, no. 6, p. e0178751, 2017.

[255] J. Zou, M. Huss, A. Abid, P. Mohammadi, A. Torkamani, and A. Telenti, "A primer on deep learning in genomics," *Nature genetics*, vol. 51, no. 1, pp. 12–18, 2019.

[256] A. Lal, Z. D. Chiang, N. Yakovenko, F. M. Duarte, J. Israeli, and J. D. Buenrostro, "Atacworks: A deep convolutional neural network toolkit for epigenomics," *bioRxiv*, p. 829481, 2020.

[257] G. Singh, M. Alser, A. Khodamoradi, K. Denolf, C. Firtina, M. B. Cavlak, H. Corporaal, and O. Mutlu, "A framework for designing efficient deep learning-based genomic basecallers," *bioRxiv*, 2022.

[258] K. Koliogeorgi, D. Mylonakis, S. Xydis, and D. Soudris, "High level synthesis acceleration of change detection in multi-temporal high resolution sentinel-2 satellite images," in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6, IEEE, 2022.

[259] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[260] K. Siau and W. Wang, "Building trust in artificial intelligence, machine learning, and robotics," *Cutter business technology journal*, vol. 31, no. 2, pp. 47–53, 2018.

[261] T. Panch, P. Szolovits, and R. Atun, "Artificial intelligence, machine learning and health systems," *Journal of global health*, vol. 8, no. 2, 2018.

[262] E. Brynjolfsson, T. Mitchell, and D. Rock, "What can machines learn, and what does it mean for occupations and the economy?," in *AEA Papers and Proceedings*, vol. 108, pp. 43–47, 2018.

[263] A. Y. Sun and B. R. Scanlon, "How can big data and machine learning benefit environment and water management: a survey of methods, applications, and future directions," *Environmental Research Letters*, vol. 14, no. 7, p. 073001, 2019.

[264] C. Gómez, J. C. White, and M. A. Wulder, "Optical remotely sensed time series data for land cover classification: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 116, pp. 55–72, 2016.

[265] J. Verbesselt, R. Hyndman, A. Zeileis, and D. Culvenor, "Phenological change detection while accounting for abrupt and gradual trends in satellite image time series," *Remote Sensing of Environment*, vol. 114, no. 12, pp. 2970–2980, 2010.

[266] L. Wu, Z. Zhang, Y. Wang, and Q. Liu, "A segmentation based change detection method for high resolution remote sensing image," in *Chinese Conference on Pattern Recognition*, pp. 314–324, Springer, 2014.

[267] S. Saha, F. Bovolo, and L. Bruzzone, "Unsupervised deep change vector analysis for multiple-change detection in vhr images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 6, pp. 3677–3693, 2019.

[268] T. Celik, "Multiscale change detection in multitemporal satellite images," *IEEE Geoscience and Remote Sensing Letters*, vol. 6, no. 4, pp. 820–824, 2009.

[269] P. Nevavuori, N. Narra, P. Linna, and T. Lipping, "Crop yield prediction using multitemporal uav data and spatio-temporal deep learning models," *Remote Sensing*, vol. 12, no. 23, p. 4000, 2020.

[270] Y. Guo, X. Jia, and D. Paull, "Effective sequential classifier training for svm-based multitemporal remote sensing image classification," *IEEE Transactions on Image Processing*, vol. 27, no. 6, pp. 3036–3048, 2018.

[271] S. Saha, L. Mou, C. Qiu, X. X. Zhu, F. Bovolo, and L. Bruzzone, "Unsupervised deep joint segmentation of multitemporal high-resolution images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 12, pp. 8780–8792, 2020.

[272] D. Ghimire, D. Kil, and S.-h. Kim, "A survey on efficient convolutional neural networks and hardware acceleration," *Electronics*, vol. 11, no. 6, p. 945, 2022.

[273] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[274] D. Wang, K. Xu, and D. Jiang, "Pipecnn: An opencl-based open-source fpga accelerator for convolution neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*, pp. 279–282, IEEE, 2017.

[275] X. Xu and B. Liu, "Fclnn: A flexible framework for fast cnn prototyping on fpga with opencl and caffe," in *2018 International Conference on Field-Programmable Technology (FPT)*, pp. 238–241, IEEE, 2018.

Appendix A.

Appendix: FPGA Acceleration of Multi-Temporal Change Detection on High Resolution Images

Machine learning tools are at the spotlight of research and human scientific activities that perform image processing and object detection. The Earth observation domain in particular heavily relies on change detection on images employing image segmentation techniques and a variety of prediction models. The computational intensity in this case lies in performing consecutive predictions for a huge incoming number of input samples. In this paper, we focus on such an application, that performs change detection on multi-temporal Sentinel-2 satellite images. The goal of this work is to explore High Level Synthesis capabilities of Intel OpenCL SDK to produce an efficient architecture for accelerating the applications focusing on optimization of a single prediction while taking into account the fragmentation of the problem. Our two-level approach first employs built-in optimization techniques to impact microarchitectural attributes and then scales this baseline to leverage coarse-grain and fine-grained parallelism. The result for the fastest implementation we acquire is a speedup of $\times 7.14$ *over the Python-TF2 implementation. This work has been published in [258].*

A.1. Introduction and Background Information

In recent years, machine learning has been established as one of the most promising research domains leading to constant advances and developments. Machine learning frameworks and neural models have been successfully applied to fields such as medicine, biology, commerce, robotics etc [259], [260], performing mainly tasks of image classification and object detection. These applications are critical for the environmental, health, social and economic domain [261], [262], [263] and therefore, scientists and researchers from both industry and academia make great efforts to develop efficient approaches and techniques for their implementation.

A special category of machine learning applications that has recently showed great progress is multi-temporal prediction on segmented images. This specifically applies to the Earth Observation domain as the need arises to locate changes, classify and detect objects on images of a certain geographical area. Thanks to advancements in the technology for remote sensing data it is now possible to acquire images of the same field of view in different chronological points in time and extract information for possible changes [264], [265]. For example, sensors such as Sentinel-2, Pleiades, Quickbird, and Gaofen-2 were recently launched and caused an explosion in the availability of high/very-high-resolution images. This has naturally led to a fast-developing research area that develops techniques and approaches to efficiently extract value out of the spatial data. Several studies of those stand out, as they indicate that it is highly effective to perform segmentation to the available images before applying a neural network for inference [266], [267].

There are multiple works that leverage the segmentation technique to perform change detection on spatial images. Depending on the type of image and the requirements of the application, each one is based on a different type of neural network operating on the image segments. The authors in [268] present a framework that performs change detection in multi-temporal satellite images. The framework generates the difference image between two time-consecutive images and decomposes it into scales using the Undecimated Discrete Wavelet Transform in order to extract feature vectors. These vectors are then assigned to a class based on k-means clustering algorithm. The authors in [269] on the other hand merge CNNs and LSTMs into a custom CNN-LSTM that helps predict crop yield based on RGB images acquired by Unmanned aerial vehicles in combination with weather data. The CNN-LSTM operates on smaller frames of the original images, aiding to better model intra-field yield variability. In [270], the proposed framework operates on a subset of the available pixels and leverages an SVM classifier which is then fine-tuned by the subsequent images. Lastly, [271] propose a deep unsupervised multitemporal segmentation method which also leverages adding semantic label to each pixel.

As a result of the segmentation process, these applications perform an immense number of inference tasks, which stem from the segmentation of high resolution images. This stresses the compute systems and delays the response time for a time series of images. The computational bottleneck is due to accumulative prediction operations rather than the complexity of a single inference task. A straightforward approach would target optimization of the core inference task. The application could of course benefit overall from such an optimization, however exhausting the resources in that direction could lead to a suboptimal solution. The problem requires a more holistic and synergetic approach that optimizes the neural network while taking into account the design architecture and parallelism across the multiple tasks.

The acceleration of the machine learning models that will be the base of our effort for acceleration is a well documented and explored domain. Survey [272] covers a wide range of accelerators targeting GPU,FPGA and ASIC devices. FPGAs in particular bring forward attributes such as low power consumption, reconfigurability and an inherent suitability for implementing efficiently multiply-and-accumulate (MAC) operations, which are dominant in both fully connected and convolution layers. The available accelerators could be categorized in three major types based on the programming model they follow. *HDL* accelerators are usually based on systolic array architectures that comprise of Processing elements. For example, Eyeriss [273] framework proposes a spatial architecture that consists of processing elements arranged in a way that exploits data reuse of filter weights and feature map and minimizes data movement of partial sums. *Highl Level synthesis* accelerators are developed in a language such as C,C++ and translated to RTL using a framework like Vivado HLS, Legup, OpenCL etc. and built-in optimization directives of the tools. In [274] the authors propose PipeCNN, a framework to accelerate neural networks by providing templated implementations of various layers of a typical NN such as pooling, convolutional and fully connected written in OpenCL. In [275], authors propose a co-design solution that integrates templated convolutional engines within Caffe. Their OpenCL implementation is based on a streaming architecture combined with a novel slicing strategy while on-chip and off-chip communication is also optimized.

In this work, we study an application of multi-temporal prediction on the Earth observation domain. The examined workflow observes changes of a designated area over a period of time by performing change detection on segmented pairs of time-consecutive Sentinel-2 data products. The generated output of the workflow is a map that includes the points where changes have occured. The main objective is to deliver an optimized solution for the application based on a stratified approach. The first level of optimization is achieved through automatic OpenCL tuning knobs and manual algorithmic changes targeting the single task of prediction on one image segment. The second level of optimization explores coarse-grained and fine-grained architectural optimizations when scaling out the design to accommodate for efficient prediction on the overall volume of incoming inference tasks. The result for the fastest implementation we acquire is a speedup of $\times 7.14$ over the Python-TF2 implementation.

The rest of the paper continues with a description of the change detection workflow and how it is implemented within the context of the OpenCL execution model (Section A.2). Section A.3 presents the hierarchical optimization approach to the problem and Section A.4 demonstrates the evaluation of the design and comparison with other works.

A.2. Multi-temporal Prediction on High Resolution Images

A.2.1. Change Detection on Sentinel-2 Satellite images

Change Detection Analysis Workflow: The examined workflow performs multi-temporal change detection on satellite images. Fig.A.1 illustrates the stages of the workflow. Each time the workflow executes, it operates on two consecutive satellite images stored in four-channel RGBA format. In the first step, the images are loaded and reshaped (step 1: *image load and extraction*) from a $4x10996x10996$ to a $100x10976x10976$ layout. The 100-length row corresponds to a $5x5x4$ tile of the initial image. Each image is divided into *batches* and all subsequent processing is performed on slices of $batch - size$. Each slice comprises of multiple 100 length arrays, i.e. tiles. The next step is a pre-processing step (step 2: $pre - processing$) that performs normalization on each tile, i.e. subtracts the *mean* value from each element and divides with the *standard deviation* of the new values. This is the input format to the Neural Network model that performs prediction on both tiles of the two images (step 3: *prediction*). The outputs of the two predictions are then subtracted per-element and contribute to the final change detection map. The resulting change detection map is the same size with the input images. Output pixel values however range between 0 and 1, representing the probability that a change has occurred between the acquisition time of the two images (step 4: *write output*).

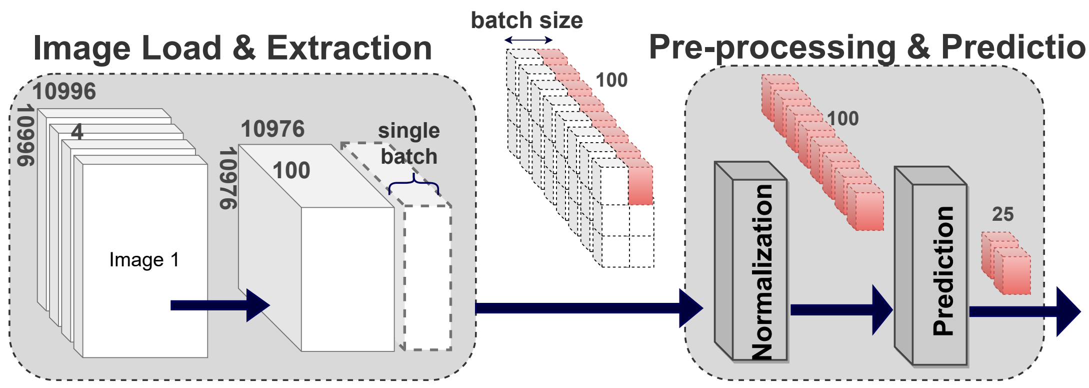


Figure A.1.: Stages of Change Detection Workflow.

Workflow Profiling: The number of high-resolution images that the workflow operates on, combined with the nature of the workflow that performs an immense number of inferences per pair of images, highlight the need for optimization and improved response time. In order to develop an efficient and effective strategy to achieve this, we first perform an elaborate profiling of the workflow. Fig.A.2 presents the distribution of execution time among distinct stages of the workflow. The *image load and extraction* as well as *write output* stages consume 26% of total time combined. The majority of execution time i.e. 74%, is taken up by the $pre - processing$ and *prediction* stages. Further analysis within these stages shows that the time is shared mainly between the normalization and prediction, with normalization being the major bottleneck. The normalization involves addition operations as well as floating point division whereas prediction includes additions and multiplications (see Section A.2.2. The division is more time consuming operation that multiplication and therefore normalization dominates the execution time.

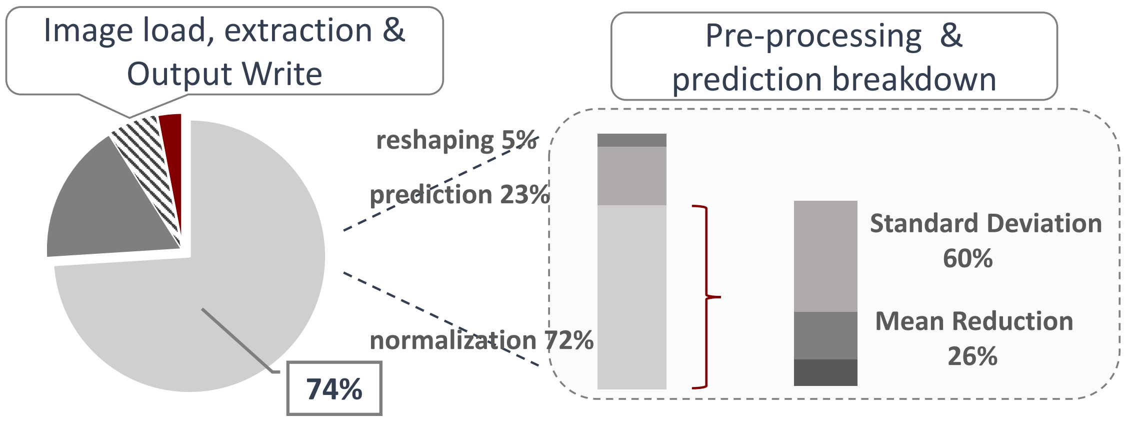


Figure A.2.: Profiling of Change Detection and breakdown of Pre-processing stage.

A.2.2. Pre-processing and Prediction Model architecture

The functionality and computational complexity of the $pre-processing$ and $prediction$ stages are explained in Listing 3 and are also illustrated in Fig.A.3. In $pre-processing$, the mean reduction is implemented through two sequential for-loops: i) the first one aggregates all elements into a single variable and ii) the second one subtracts this value from each element. Standard deviation division requires three loops that traverse the input array: i) the first one calculates the new mean value, ii) the second one computes the standard deviation , i.e. the squared root of the sum of the squared differences of each element and the mean value divided by the number of elements and iii) the last one divides all elements with standard deviation. Once the pre-processing of the tile is finished, a prediction model operates on the normalized data. Fig.A.3 depicts the architecture, i.e. the layers, of the utilized prediction model. As mentioned, the image tiles are already flattened into arrays of length 100. A *Dense* layer receives this input and outputs a vector of length 25. The Dense layer has a weights matrix that comprises of 25 rows of 100 elements. Each element of the output vector is the result of a dot-product operation between the sample vector of length 100 and a row of the weight matrix. This matrix has 25 rows of length 100. A bias value is added to each output element and an activation function is applied on it, i.e. Leaky ReLU. Lastly, a $L2-normalization$ layer operates on the output vector by dividing each element with the squared root of the sum of the squared elements.

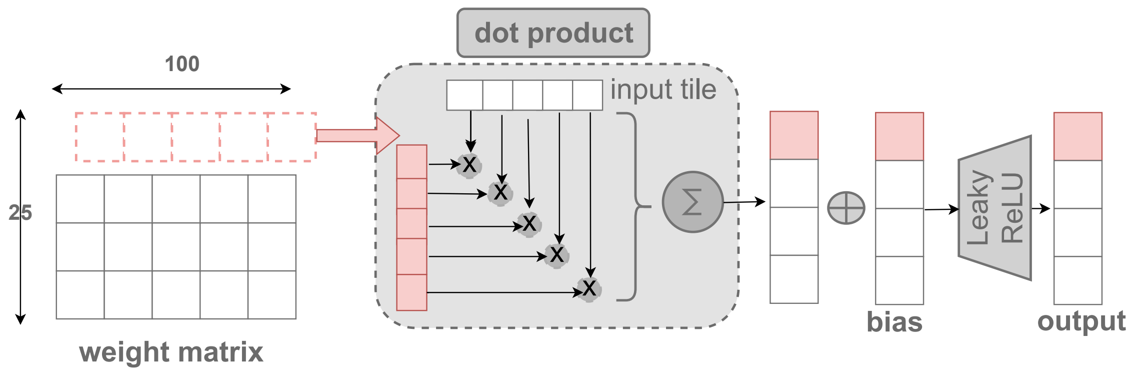


Figure A.3.: Architecture of Prediction Model for Change Detection

Algorithm 3: Normalization and Prediction of Tile

```
1  const N = 100; const M = 25; //Mean reduction
2  subtract_mean_value() //2*O(N)
3  //Divide with standard deviation
4  compute_std_deviation() //2*O(N)
5  divide_with_std_deviation() //O(N)
6  //Dense Layer O(M*N)
7  for 1 to M do
8      for 1 to N do
9          dot_product()
10     add_bias()
11     Leaky_ReLU()
12 //L2 Normalization
13 sqrt = sqrt_of_sum_of_squared_elements() //O(N)
14 divide_with_sqrt //O(N)
```

A.2.3. Pre-processing and Prediction as an OpenCL kernel

The $pre-processing$ and $prediction$ stages as described in Listing 3 constitute the major bottleneck and are going to be implemented as an FPGA accelerator through the OpenCL programming model. OpenCLs' main component is the kernel. A kernel is issued through a host code and scheduled to execute within a command queue that is necessary for host-device interaction. The source code described within the kernel corresponds to a single $work-item$. Upon runtime, the host code can request for a specific number of $work-items$ to execute, i.e. *global size*. The programmer can also specify the *local size*, which is the number of $work-items$ that should be grouped into a $work-group$. Depending on the OpenCL implementation the $work-items$ within a single $work-group$ could execute concurrently but in most cases they execute sequentially.

$Work - groups$ can execute concurrently based on the number of compute units in the device. In any case for a $global\ size$ and $local\ size$ of 1, the configuration is equivalent to a single work group that includes only one work item that executes the source code inside the kernel. In our case, the $pre - processing$ and $prediction$ stage are described within the same OpenCL kernel. Therefore, a single $work - item$ implements the normalization and prediction of a tile of an image reshaped into an array of length 100. Increasing the $global\ size$ leads to normalization and prediction of more tiles, i.e. a larger image part. This straightforward implementation corresponds to an unoptimized accelerator that does not include any OpenCL tuning knobs and adopts the default memory configuration, e.g. all read-write operations access the global memory.

A.3. Hierarchical Optimization Strategy of Model Architecture

This section describes the optimization methodology that is adopted in order to effectively examine the available design space and generate efficient architectures for change detection in multi-temporal images. First, we explore the native OpenCL capabilities to tune the design on a micro-architectural and memory hierarchy level and derive an efficient baseline architecture. Followingly, we build upon this architecture and leverage the coarse grain parallelism available through the OpenCL computing model tofurther enhance the design for execution on multiple image segments. Simultaneously, we explore alternative architectures that focus on fine-grained parallelism.

A.3.1. Optimizing Baseline Architecture

This section elaborates on the techniques that secure an initial performance enhancement of the single-task kernel. The goal is to reach the optimized architecture illustrated in Fig.A.4, which exploits parallelism within each computational core of pre-processing and prediction stages and benefits from optimized data transfer and memory interconnections.

OpenCL native tuning knobs

A first level of optimization can be achieved through meticulous tuning of OpenCL built-in directives. The most efficient and commonly used directive in HLS tools is loop unrolling. In OpenCL loop unroll is indicated by including *#pragma unroll* followed by an unroll factor (partial or full) on top of the for loop in the source code. In our case, loop unrolling can be applied on all loops. For mean reduction, standard deviation and L2-normalization

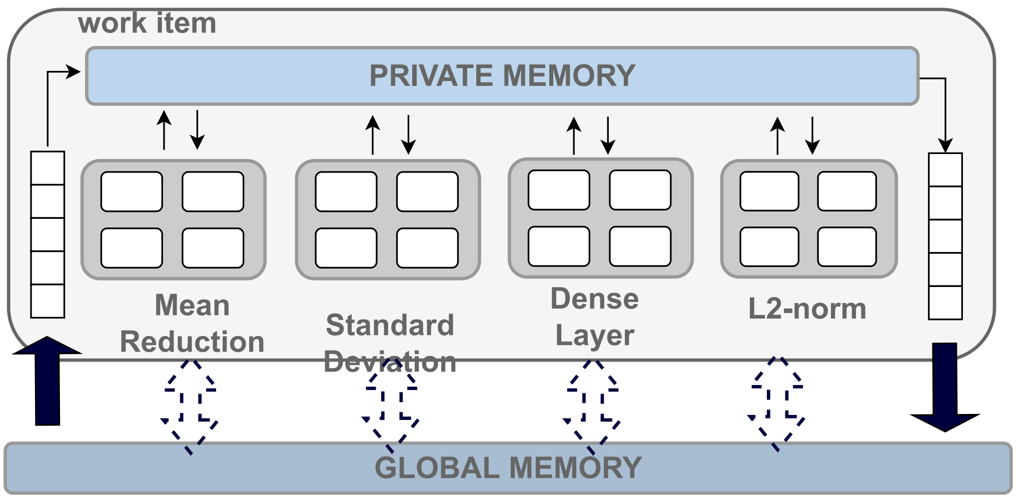


Figure A.4.: Overview of Design with Microarchitectural Optimizations.

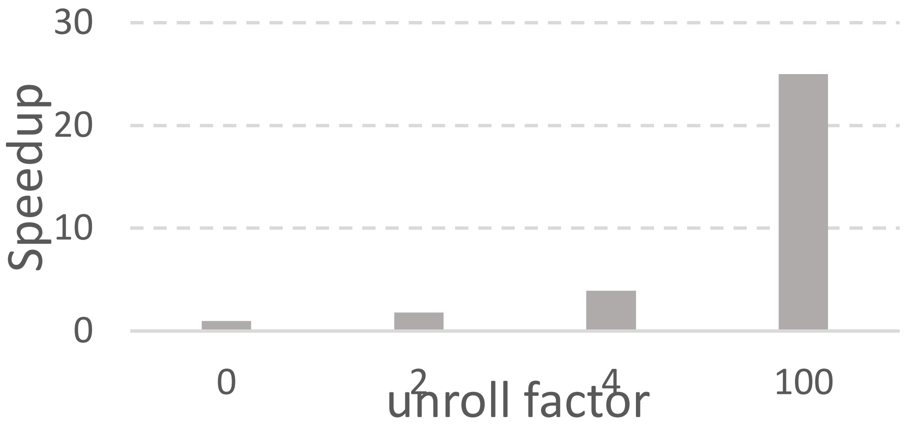
each operation operate on individual elements and there are no data dependencies transfered across iterations. Regarding the Dense layer, unrolling can be applied on both the inner and outer loop. For the inner loop, unrolling parallelizes the dot product operations, i.e. the multiplications illustrated in Fig.A.3, as there are no *read after write* dependencies. For the outer loop, unrolling paralellizes the dot product computation of the same input with different rows of the weight matrix.

Fig.A.6 illustrates how unrolling *mean reduction*, *standard deviation*, $L2-normalization$ and the $dot-product$ loops by the same factor impacts the latency and resources of the design. The logic utilization reduces as unrolling a loop extinguishes the loop control overheads. DSP utilization escalates as more operations can now be scheduled in parallel. For smaller unroll factors, the speedup scales linearly and proportionally to the unroll factor. However, for full unroll the speedup does not follow the linear trend and is restricted to a $\times 25$ speedup for an unroll factor of 100. This fact in combination with the considerable increase in BRAM utilization, especially for the case of full unroll, indicates that the memory configuration may cause a bottleneck. We examine this scenario in the next Section.

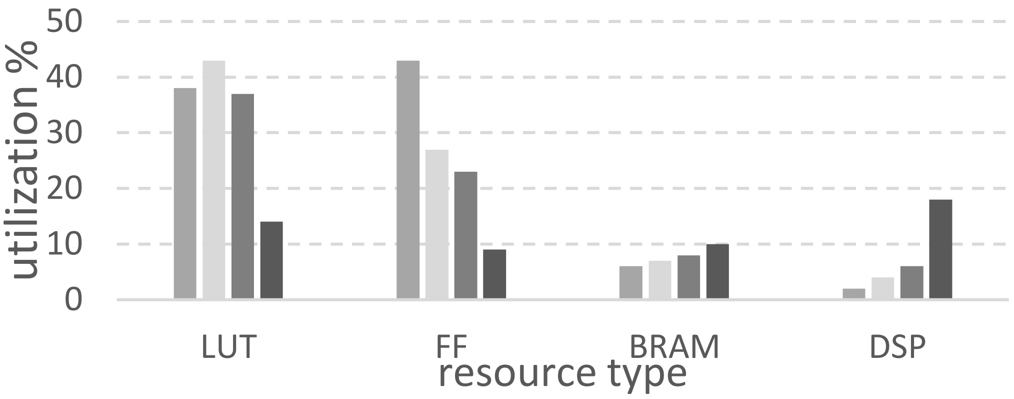
Memory architecture optimization in OpenCL

Memory architecture can be optimized both in terms of hierarchy and data layout and it greatly impacts the overall performance.

Memory hierarchy configuration: Memory hierarchy configuration is critical for generating an efficient RTL architecture, as it affects global interconnections. In our imple-



(a) Speedup for unroll factors 2,4 and fully unroll.



(b) Area utilization for unroll factors 2,4 and fully unroll.

Figure A.5.: Speedup and Resources utilization when applying unroll for various factors.

mentation, the weight matrix and bias array are implemented as constant variables and are therefore hardcoded , whereas the input and output arrays (length 100 and 25 respectively) as arguments are by default implemented as buffers on global memory. All intermediate operations are executed on the input array and therefore all read-write operations have to access the global memory. In particular, a total of 12 global interconnections are formed: i) two for reading and one storing the mean-reducted data, ii) four for the deviation-division part, iii) two for reading the data and weights in dense layer and one storing back, and lastly iv) two for reading and normalizing the output values.

However, high global interconnections result in increased resource usage and lower performance. In order to alleviate this overhead, we utilize the private address space. This is realised by copying the input array into an array declared as *__private*. The compiler implements this as registers whenever possible and this leads to very efficient hardware, performance and resource-wise. Eradicating global interconnections by moving the computation on the private memory as shown in Fig.A.4, improves the performance and most importantly drops resources utilization in half (Fig.**??**).

Memory layout: Memory layout and data placement can increase ports and locality and in turn increase performance. In this section, we will examine two different memory partitioning schemes and their impact in the overall design efficiency.

Cyclic Partitioning: Cyclic partitioning transforms a single array into multiple smaller arrays so that each partition contains consecutive elements of the initial arrays allowing concurrent access to them. In our case, this scenario takes places when unrolling the for-loops as described in Section A.3.1. We will demonstrate cyclic partitioning on the inner loop of the Dense layer that computes the dot product between two arrays. As it is illustrated in Fig.A.3, the elements of the input array are multiplied with the same order elements of a single row of the weight matrix. These multiplications are independent from each other and can execute in parallel. Unrolling the loop allows for parallel execution of the multiplications, however this is hindered by the limited read ports available in both arrays (i.e. 2 ports per BRAM). OpenCL however does not provide directives that perform memory partitioning and reshaping automatically. To mitigate this, we declare multiple partitions both for the input array and weight-matrix and manually unroll the loop so that in each iteration the different partitions are explicitly accessed. The number of partitions depend on the unroll factor. Fig.A.7 shows the positioning of the elements in the correct partitions to facilitate parallel access.

TThis code restructuring is further customized and tuned to explicitly guide the compiler to perform the accumulation of the products in a tree-like manner. For that purpose we utilize temporary variables in order to store intermediate sums. This explicit coding style helps the compiler infer the need to allocate more hardware resources, i.e. adders, multipliers. The use of command line flag *-fp-relaxed* allows the compiler to schedule the independent operations in parallel, so that they exploit the extra hardware, and transform the graph of the operations in the tree-like structure depicted in Fig.A.8.

BUS Bandwidth Optimization: An alternative approach is to explore potential advantages to increasing the data locality and bus width instead of the memory ports. For that purpose, we reshape the matrix so that sequential elements that get accessed within the same iteration of the unrolled loop are grouped together in a single struct. The array is declared as an array of structs and each struct contains an array of $unroll - factor$ elements. Fig.A.7 illustrates the new data struct and the placement of the original data in the expected order. As a result of this layout, the data bus width is greater and the elements that are required in parallel are fetched together. This behavior emulates the memory reshaping technique that groups multiple elements in a single element with greater bitwidth.

Arithmetic and Low-level microarchitectural tuning

Careful study of the OpenCL generated reports revealed high area utilization due to division operations in the source code. Multiplication on the other hand consumes less resources. Therefore, we modify the source code and replace all division operations with *value* with multiplication with the reverse $\frac{1}{value}$. Specifically, this code optimization has been applied to the division operation of the input with the standard deviation value and to the L2 Normalization on the output array. A profiling has been implemented to ensure that the propagation of the different floating point value in the pipeline does not decrease the accuracy of the final results. The results show a reduction in LUT utilization by 75% and a minor performance improvement by a **2.4%**. This reduction in area utilization is crucial when scaling a design and perform other optimization techniques to increase performance.

As a last insight, it is interesting to show how the storing of weight matrix impacts the design. Instead of declaring the weights as a kernel parameter, we opt instead to declare the matrix as a *constant* and therefore statically allocate the matrix. BRAM utilization increases from 29 to 58% whereas DSP and FF utilization are not affected. The execution time however improves by 22%.

A.3.2. Optimizing horizontal and vertical scaling

In Section A.3.1 we examined ways to optimize the work of a single work-item. The performance can be further improved by scaling the design. In this section we explore two options for achieving that, horizontallly and vertically.

Latency-Optimized Architecture

The first architectural approach explores horizontal scaling, as it replicates the hardware instantiated to ensure parallel execution of the kernel. This is essentially based on the principle of coarse-grained parallelism, i.e. distributing the workload to multiple workers, each one performing the same task and all of them working in parallel. OpenCL provides two alternatives to achieve this, i) replicating the compute kernels on hardware and ii) kernel vectorization.

The first approach is implemented through the **num_compute_units** directive which specifies the number of times that the kernel is replicated inside the device. It is included in the source code above the kernel of interest. Each kernel compute unit can execute multiple work-groups/items simultaneously. During the run-time the OpenCL environment dynamically distributes the work groups/items across the compute units. Fig.A.9

illustrates an example of how multiple work groups get scheduled to execute on each of the four allocated kernels on the device.

An alternative directive that instantiates more hardware is *num_simd_work_items*. This directive increases the number of operations executed per work-item by vectorizing the kernel. Kernel vectorization allows multiple work-items to execute in a single instruction multiple data (SIMD) manner. Each work item now performs more work that scales linearly with the number of SIMD lanes. This architectural approach is illustrated in Fig.A.10 for a 4-lane SIMD architecture. The dashed schema shows the combination of the two approaches for two compute units and 4-SIMD lanes.

Throughput-Optimized Architecture

In contrast with Section A.3.2, the technique described in this section focuses on vertical scaling, by creating a pipeline and therefore an efficient fine-grained architecture. The goal is to break down the workflow into distinct smaller tasks that can operate in parallel and communicate or share results with each other when necessary. Such an architecture consumes less resources and optimizes throughput rather than the latency of generating a single output.

The mechanism which provides the ability for data to be shared among various kernels is using ***OpenCL Channels***. The first step in applying this technique is to break down the initial kernel in multiple smaller kernels. Channels are FIFO buffers, that allow the smaller kernels to communicate directly with each other with high efficiency and low latency, following a $producer - consumer$ relationship. In our case we transform the $pre - processing\ and\ prediction$ so that it follows a pipelined execution model. The pipeline is comprised of the following stages: i) mean reduction, ii) standard deviation division, iii) dense layer and iv) L2 normalization. Each one of these stages is declared as a separate kernel. Each distinct kernel calculates a part of the pipeline and writes the results to a buffer-channel. The channel is then read by the next kernel, and so on, up to the point where data are written back to memory.

Initial profiling reveals that latency is not equally distributed across the stages, causing a bottleneck. Specifically, mean reduction kernel is slower due to loading the images from global memory rather than a channel. To alleviate this latency overhead, we add an $read - input\ kernel$ as well as a $buffer$, as depicted in the $single - pipe\ buffered$ architecture in Fig.A.11. The input kernel reads data from global memory and writes them to a channel whereas the buffer kernel propagates them further on. This way the wait time is absorbed and the mean reduction kernel reads and produces data with the same rate as the other kernels.

Similar to the latency-optimized architecture, this fine-grained architecture can be scaled horizontally. By default, OpenCL does not support the employment of channels and kernel vectorization at the same time. To mitigitate this issue we explicitly replicate all kernels and channels to reach FPGA capacity and create therefore a $double-pipe\ buffered$ architecture.

A.4. Experimental Evaluation

A.4.1. Experimental Setup

The proposed accelerator runs inside a docker container that is set up with all necessary software requirements and python libraries of the original Change Detection tool. The host machine is a 14-core Intel Xeon Gold 6132 with 132GB of DDR4 memory that is connected through PCIe with an Intel Stratix 10 FPGA device. The utilized data are real Satellite Sentinel-2 images.

A.4.2. Performance evaluation

This section studies the impact of various parameters in the performance of the scaled architectures presented in Section A.3.2. The coarse-grained and fine-grained architectures described are build upon one of the baseline optimized microarchitectures presented in Section A.3.1. The selected configuration is a result of a greedy approach that iteratively applies each optimization technique and in every step chooses the configuration that boosts performance and keeps utilization relatively low. It includes: i) utilization of private memory, ii) loops (except the outer loop of dense layer) are manually unrolled by a factor of 4 and arrays sequentially partitioned by the same factor, iii) each loop is further annotated with the $pragma unroll$ directive, iv) division operations are replaced with multiplications, v) *-fp-relaxed* option is enabled to assist in the balancing of the operations and vi) the weight matrix is declared as a constant matrix for static allocation.

Latency-optimized designs

Fig.A.12 present the performance and resources utilization of designs with multiple compute units of the baseline optimized kernel for an increasing number of $global\ size$. This translates to more work items being scheduled, each one operating on a single tile of the image. For 2 compute units and as the problem size grows, the execution time drops in half. This is not the case for compute units 3 and 4, as their performance stays close to that of 2 compute units. The reason for that is that kernel replication leads to a linear scaling in global interconnects and increased wiring utilization and finally a drop in maximum possible frequency. Indeed, for four compute units, there is 39% drop in frequency of the kernels.

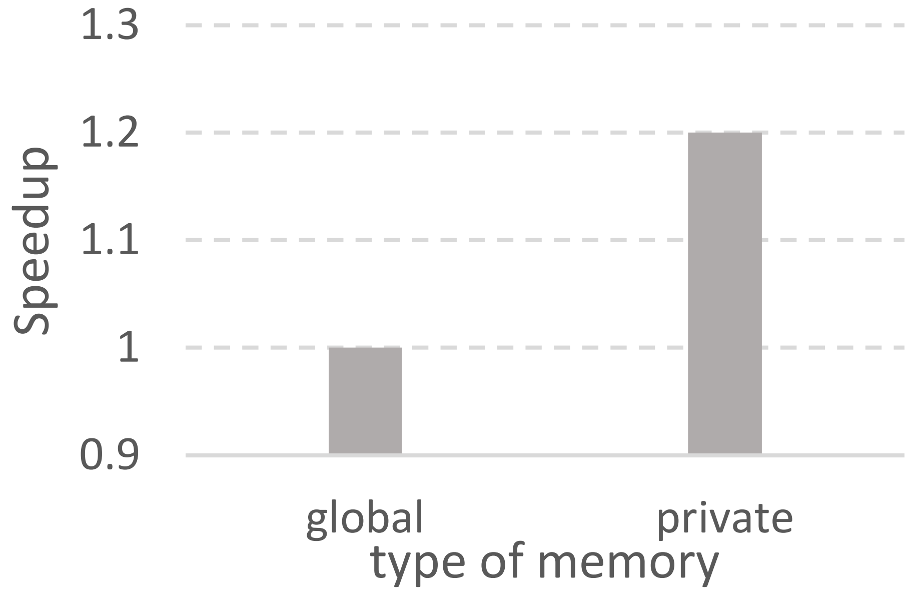
The corresponding results for the Single Instruction Multiple Data method are presented in Fig.A.13. In this case, increased wiring leads to a drop in frequency from 295MHz for two-lane SIMD to 219MHz for four-lane SIMD. Therefore the $\times 2$ speedup acquired for two-lanes, is not doubled when moving to four-lanes but enhanced by a mere additional **10%**.

Throughput-optimized designs

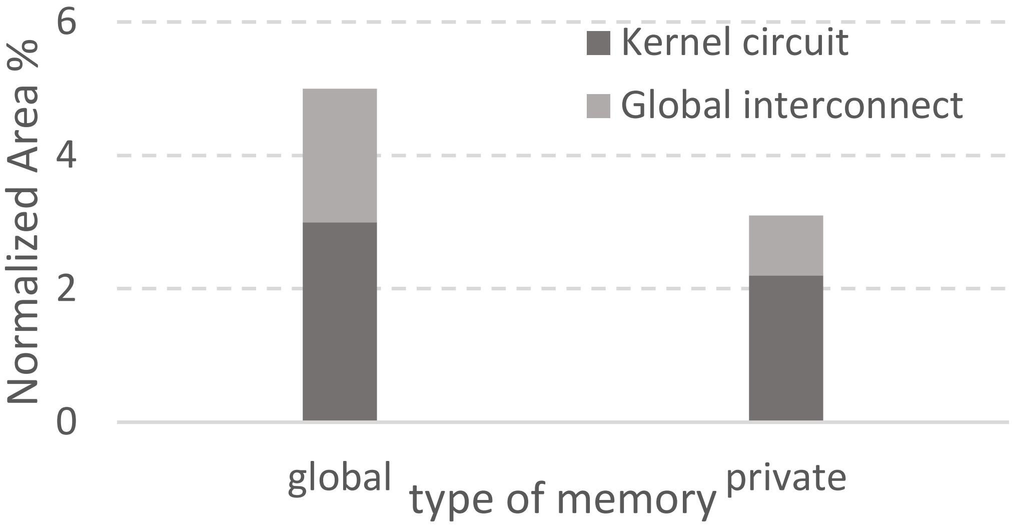
The three fine grained architectures presented in Section A.3.2 are also evaluated in terms of performance and resource utilization for an increasing number of input tiles. As seen in Fig.A.14 the $single-pipebuffered$ architecture speeds up the design by about **35%** without consuming more resources, whereas the $double-pipebuffered$ architecture results in a **65%** increase in performance. Comparison with Latency-optimized builds, indicates latency optimized designs score higher performance for a small number of threads, however as thread size increases, throughput based architectures are gradually closing the gap and even outperform the latency optimized ones.

Comparison with software

In this part, we will compare the fastest implementation that resulted from each one of the coarse-grained and fine-grained architectures to the python pre-processing and the built-in predict method that is utilized in Tensorflow v2. The fastest coarse-grained architecture includes three compute units and the fastest fine-grained architecture is arranged in a $double-pipe\ buffered$ pipeline. The predict method of Tensorflow internally is also optimized for prediction in batch mode. Fig.A.15 shows that the coarse-grained architecture is $\times 7.14$ faster than the software implementation whereas the speedup for the fine-grained is $\times 5.79$ when the input is a single satellite image of 12GB.



(a) Speedup acquired when utilizing private memory for intermediate operations.



(b) Area utilization for unroll factors 2,4 and fully unroll.

Figure A.6.: Decrease both in kernel and interconnect logic when utilizing private memory for intermediate operations.

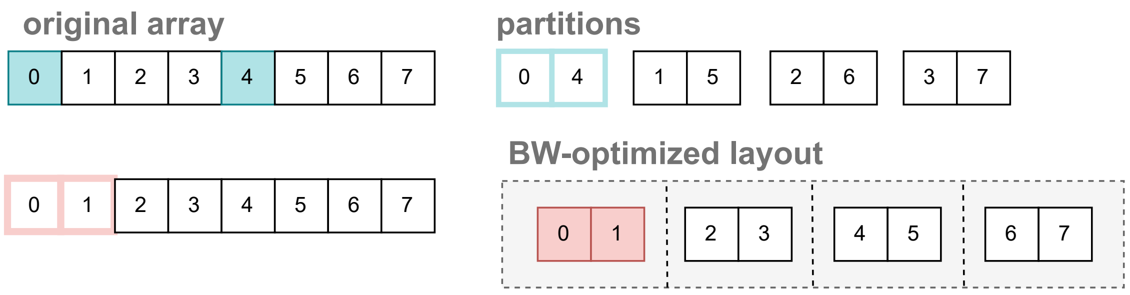
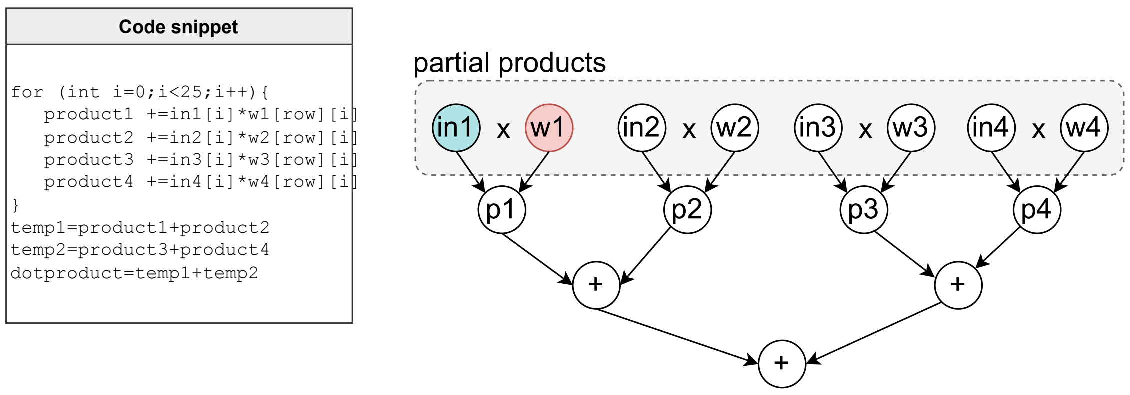


Figure A.7.: Memory optimization schemes for parallel access to data: i) Cyclic partitioning for unroll factor 4. ii) Sequential partitioning and bus BW optimization for unroll factor 4.


Code snippet

```
for (int i=0;i<25;i++){
   product1 +=in1[i]*w1[row][i]
   product2 +=in2[i]*w2[row][i]
   product3 +=in3[i]*w3[row][i]
   product4 +=in4[i]*w4[row][i]
}
temp1=product1+product2
temp2=product3+product4
dotproduct=temp1+temp2
```

Figure A.8.: Graph for Tree-balanced operations in unrolled dot-product loop.

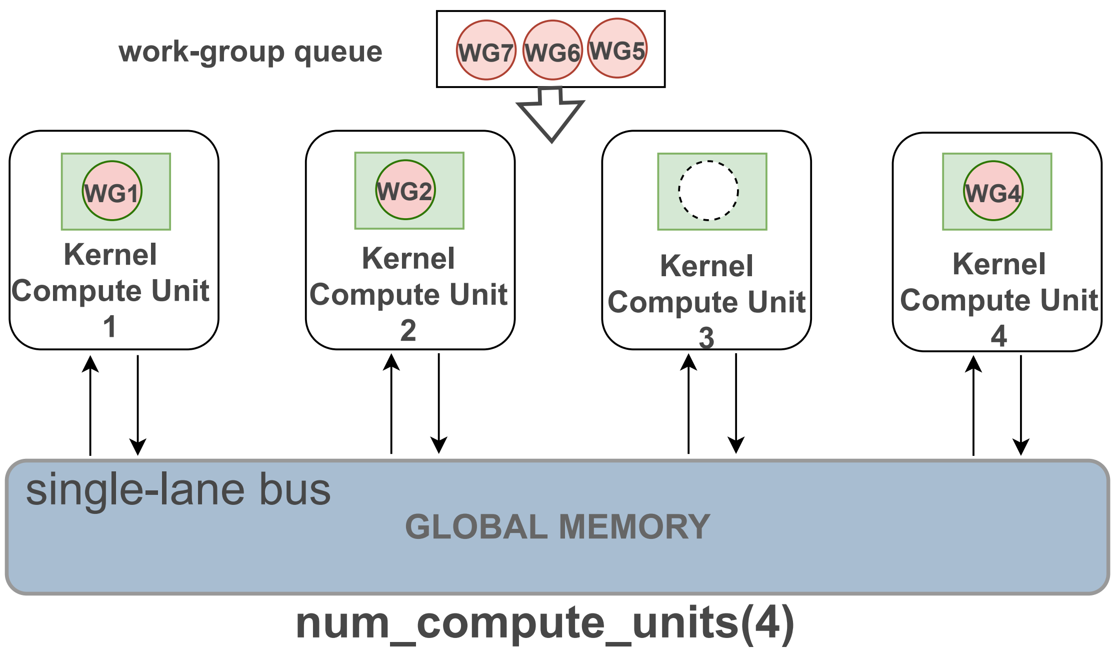


Figure A.9.: Kernel replication with 4 kernel instances. Work-groups are scheduled onto the available compute units.

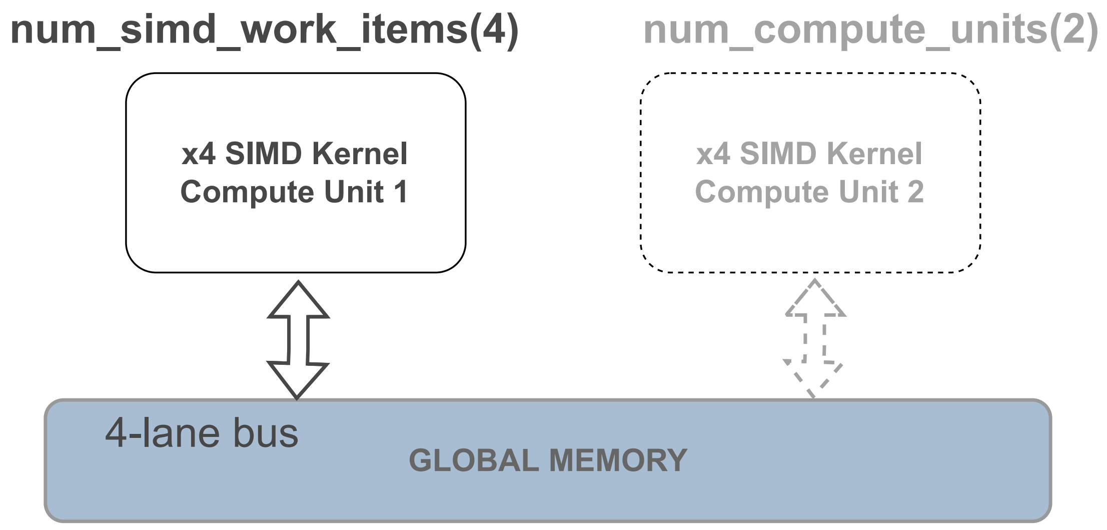


Figure A.10.: Kernel vectorization by a factor of 4. The technique can be combined with multiple compute units, e.g. 2.

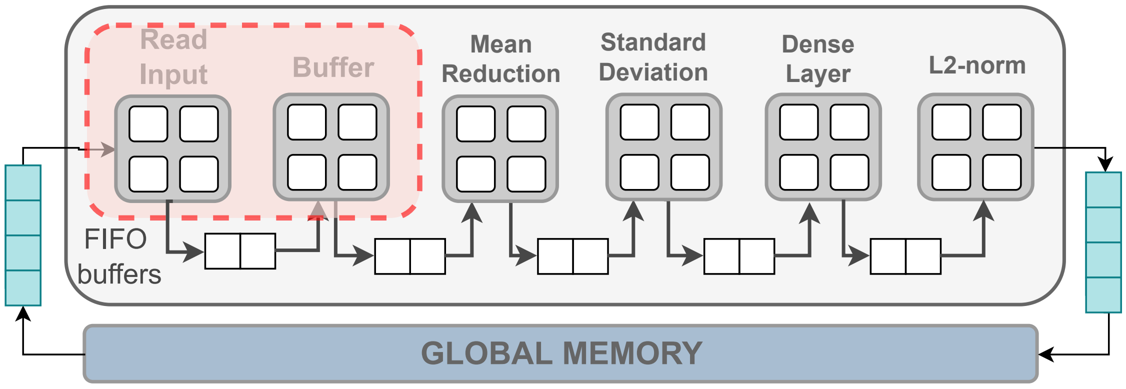
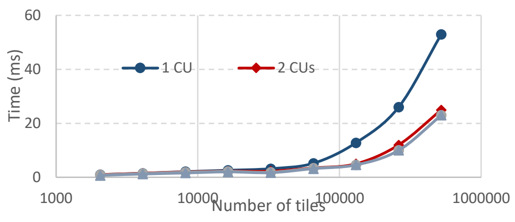
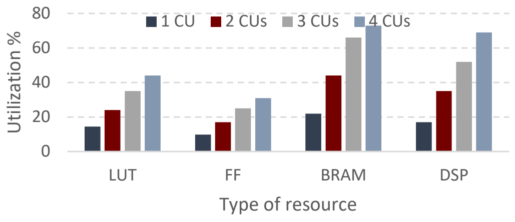


Figure A.11.: Extended pipeline with *input* and *buffer* stages to balance the latency of each stage.

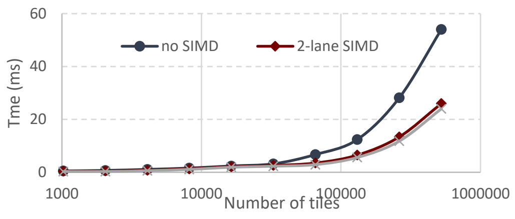


(a) Execution time for multiple compute units and increasing dataset.

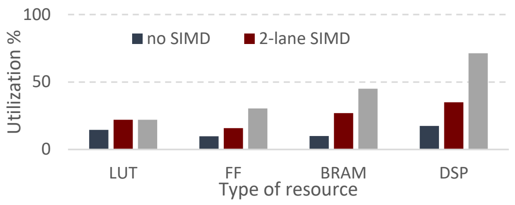


(b) Resources utilization for multiple compute units.

Figure A.12.: Scaling the design with multiple compute units.



(a) Execution time for increased vectorization factor and increasing dataset.



(b) Resources utilization for increased vectorization factor.

Figure A.13.: Scaling the design through vectorization.

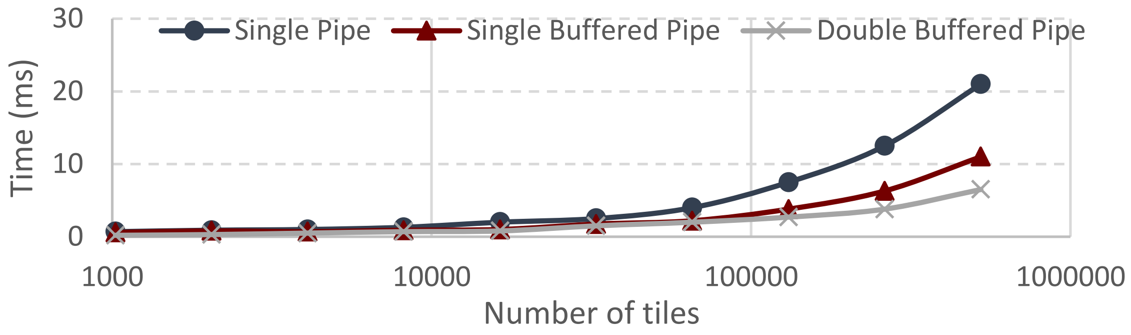


Figure A.14.: Execution time for examined fine-grained architectures.

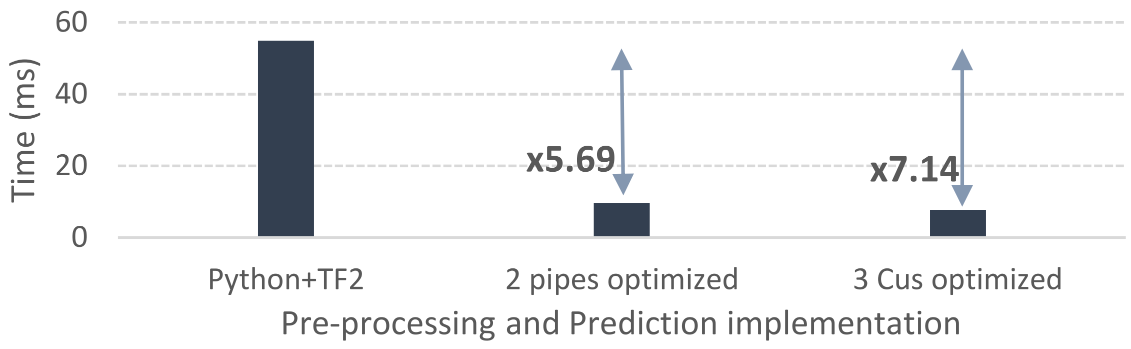


Figure A.15.: Comparison of execution between best coarse and fine-grained accelerators and Python-TF2 implementation.