UNIVERSITY OF TALCA
ENGINEERING FACULTY
CIVIL COMPUTER ENGINEERING SCHOOL

Hardware-accelerated similarity search index implementation on a Adapteva Parallella-16 reconfigurable computing machine

ERIK REGLA

Advisor: RODRIGO PAREDES

Memoria para optar al título de Ingeniero Civil en Computación

Curicó – Chile
month, year

UNIVERSITY OF TALCA
ENGINEERING FACULTY
CIVIL COMPUTER ENGINEERING SCHOOL

Hardware-accelerated similarity search index implementation on a Adapteva Parallella-16 reconfigurable computing machine

ERIK REGLA

Advisor: RODRIGO PAREDES

Profesor Informante: PROFESOR INFORMANTE 1

Profesor Informante: PROFESOR INFORMANTE 2

Memoria para optar al título de
Ingeniero Civil en Computación

This document was graded with a score of: _________

Curicó – Chile

month, year

Dedicated to the me of the future.

ACKNOWLEDGEMENTS

Acknowledgements to ...

CONTENT INDEX

FIGURE INDEX

TABLE INDEX

SUMMARY

Here goes the abstract

1. Introduction

1.1 Context

Currently developers can enjoy of a vast availability of computational resources to perform many resource-hungry tasks, such as real-time load balancing, molecular simulations with fairly accurate results, etcetera. But at the same time computer scientists are struggling with limits of hardware, more specifically, transistor size.

With the current advances in transistor technology, the design of 3D layered silicon chips and 5nm lithography, Moore's Law is now a problem as now is harder to push the hardware limits -for single chips- at the same rate as ten years ago[**?**, **?**]. The Altera Corporation stated in 2007 in a technical report that: *"For most of the microprocessor's history, application demands have risen in response to processor improvements, allowing processors to stay ahead of demand. In the last few years, however, the situation has changed. High-performance computing (HPC) applications are now demanding more than processors alone can deliver, creating a technology gap between demand and performance."*[**?**]

The use of heterogeneous computer architectures is one of the most popular methods nowdays to tackle this *Technology Gap Problem*, solutions like Nvidia CUDA [**?**] allows developers to harness the massively parallel power of present-day GPUs to perform intensive tasks, as long as the problem fits the constraints imposed by the hardware.

As such, there is a growing interest on *high-performance and low-power custom computing machines* which aim to design *Application-specific integrated circuits* (ASICs) in order to solve certain computational problems. But as ASIC design, prototyping, and implementation is very costly, *Field Programable Gate Arrays* (FPGAs) has been proven as a cost-effective solution to implement those designs and interconnect them with our current platforms to form reconfigurable computing architectures [**?**, **?**, **?**] being only limited by their power-budget while offering a scalable compute model at the same time as overcome many of GPGPU current problems as energy consumption and performance degradation when facing intensive branch-divergence control-flow algorithms [?], posing many applications in practice, most of them in the robotics field, but also on high-performance computing.

1.1.1 Motivation

On non-traditional databases, metric spaces indices offer an abstraction to classify objects using the underlying notion of space and distance which is natural for humans to understand. Those indices work by placing elements on a hiperdimensional grid allowing to query for their closeness between objects. One of the many approaches to perform similarity searches is to perform k-nn or range queries on permutant-based indices, which abstract the dataset dimensionality and the cost of the object-object distance calculation. To perform a search on this index, a permutation is generated for the query object and then compared to the whole dataset under the premise that computing distance between two permutations is faster than computing the full distance between the two elements. After their distances are computed, a subset is selected under a certain criteria given by the query nature and the results are filtered later in order to answer the query. [?, ?]

The implementation of indices for approximate search on metric spaces pose a great problem, since as the operations involved in the distance calculations are rather easy to perform on a CPU, those operations are executed individually for each element of the index, regardless it is part of the solution or not. This behaviour makes similarity search indices a perfect test bench for its implementation on a reconfigurable computing device[?] such as the Adapteva Parallella-16, a *System on chip* (SoC) based on the Xilinx All Programmable Zynq7000 Series SoC which packs together a Dual-core 32-bit ARM Cortex-A9 host controller and a Artix-7 FPGA [?, ?]. The board also has an Epiphany III multicore accelerator coprocessor, interconnected with the host-controller though an FPGA designed ASIC in order to be used as a low-power high-performance heterogeneous computing platform.

One of the main problems with reconfigurable computing is the complexity of circuit design [?]. Independently of the algorithm being ported to an FPGA implementation, there is not automated way to use the same source code used on the Processing System (PS) version of the problem in the Programmable Logic (PL). As there are some tools developed for both FPGA manufacturers and 3rd parties to design circuits using the C language, they lack of precision and many considerations and “compile” optimizations must be performed in order to successfully port certain algorithms[**?**], such as instruction pipelines, read/write syncronization, clock gating[**?**], etc [**?**, **?**, **?**, **?**]. As such this work intends to serve as a guide to future hardware designers and developers and to demonstrate that with proper study we can port some algorithms to certain reconfigurable computing machines without major hassle.

1.1.2 Goals

Main objective

- Study the feasibility of implementing hardware-based accelerators for the Adapteva Parallella SoC.

Specific objectives

- Specify requirements and considerations to be accounted when porting general purpose algorithms to FPGAs.
- Study and implement a data sharing sharing solution between the Programable Logic and the Programable Software.
- Develop a functional FPGA-based hardware-accelerator prototype for a subset of routines involved on approximate similarity search.
- Deliver a solid guide to serve as a starting point to future computer scientists with little or no knowledge about hardware design.

2. Methodology

2.1 Methodology

This problem required now only knowledge in computer science but also a prior background on electronics, as such, is needed to research about the following topics before going into serious development:

- **Non-traditional databases and metric spaces**
- **FPGA development**
- **High-level synthesis**
- **Interconnection between programable logic and program software**

2.2 Implementation Methodology

To develop both the experiments and the acelerator we will follow an interative development cycle, based on the results of each source code but not following a exact construction metodology because of the nature of this work. Namely the elements to iterate can be summarized as follows:

- **Permutant-based index testbench:** This implementation is intented to be used to gain insight of which methods are the most frequently executed and to use it as reference to compare the results with the acelerator-improved version. It will be developed using C++ and it will provide a graphical interface to visually debug the queries.

- **FPGA Accelerator:** This artifact it will be used to test the interconnection capabilities of hardware accelerators over the Zynq board, and it will not provide drivers for userspace memory management rather it will operate directly with each DRAM block present on the board. It will be developed using Xilinx Vivado SDK for FPGA programming and with Xilinx Vivado HLS to write C/C++ code targeted to be synthetized on the on the board as an acelerator IP.
- **Accelerated index testbench:** Once the FPGA accelerator is ready, this artifact will provide insight on how the accelerator implementation affects the performance of the original index. This implmentation will be optimized to run on ARM architectures and it will be a memory-coalesent implementation of the first index implemented at the beggining.
- **Kernel drivers:** This artifact will enable our accelerator to peform memory-safe operations by restricting memory access on userspace. It will be developed using C and designed aroung GNU Linux guidelines for kernel drivers and modules.
- **Accelerated index:** This artifact will wrap everything together, both the accelerated and non accelerated versions and it will be used to measure the final performance of the index.

As it can be seen, there is a need to iterate though every artifact of this research.

2.2.1 Design and architecture

2.2.2 Development

2.3 Testing and experimentation Methodology

For experimentation we will follow an experimental approach rather than a purely theoretical one [**?**] to devise how the accelerator affects the behaivour of the index. For such purposes a series of tests will be performed:

- **Hot zones on permutant index:** The bigger benefits of implementing a stream-like accelerator are reaped when the operation to perform is simple and it can be streamlined and paralelized. So, after implementing the index, we want to map all code zones which are in most demand and then classify by their complexity and size. This two parameters will be the base to decide which ones are feasible to implementation on the FPGA as accelerators.

- **Litography and synthesis:** As we will be using high-level synthesis to implement the accelerators, there is a need to study the effects of different `PRAGMAS` in the accelerator source code. It is expected that pragmas that increase the thoughput of our accelerator will also be in higher demmand of resources which could be not be satisfied by our entry-level development board, which also affects the scope of the accelerator covered as well as the synthesis time required to create a given IP.

- **Kernel Userspace I/O Drivers:** There are several ways to build kernel drivers and modules, some of them treat as mere files, while another approaches treat each device descriptor as text files, streams or raw memory mapping. Each one of those will have pros and cons that will be need to be studied in order to correct the implementation to be memory-safe and scalable.

- **Memory-safe implementation:** The implementation cannot go into a non-memory safe state as FPGA will have raw access to DRAM and this can suppose many problems as the index being stuck or memory corruption to other processes. This involves the implementation of checked runtime checks along the execution which are expected to be done though a correct memory management on userspace and bound checking on hardware.

3. Required knowledge

3.1 Metric Spaces and Non-Traditional Databases

3.1.1 Overview

Chávez and Navarro [**?**] gives an accurate description of what a non-traditional database is and which problems intends to solve. To summarise the whole idea of the former paper, traditional databases can be seen as engines that answer queries based on the premise that for a certain key there is a certain element with a (possibly unique) relationship. In other words, for a certain query Q with constraints C there is an answer A which satisfies $Q \times C$. Actually, there is nothing wrong with this approach, but it's really complex and expensive to execute queries were we do not desire a exact answer because it does not exists or because it is to resourse-expensive to compute, for instance, if we have a large dataset of points in the space and we want to query the "10 points more closer to a point Q" (this is also known as a k-nearest-neighbour query) we need to execute this query for each point in the database and to compute their distances as well, which we don't know beforehand how costly it is.

This is an interesting property which multimedia objects which makes pointless the use of exact queries as there is no two objects with the exact parameters -unless they are exact copies-. A "real" world example of this situation is plagiarism detection on multimedia files. Plagiarism[**?**] can be hard to detect because in most cases, only a part of the multimedia resembles the original work [**?**]. There are many approaches to this problem, but in music one of the easiest representations is the "spectrogram"[**?**] which is a visual representation of the different waveforms involved on a certain audio file. Spectrograms allows the extraction of multiple features from a single audio picece such as scale, BPM (beats per minute), movements, entropy and so on, so forth.

The values of this features can be allocated as *vectors* $\mathbb{X}$ on a *metric space* $\mathbb{U}$ formally defined as follows:

For a given cartesian pair $\mathbb{U} = (\mathbb{X}, d)$ where $\mathbb{X}$ is a non-empty set of elements and is a distance function which operates over $\mathbb{X}$ in the following manner $d : \mathbb{X} \times \mathbb{X} \rightarrow [0, \infty)$, then it is considered a *metric space* if it satisfies all the following conditions:

1. **Reflexibility:** $\forall x_1, x_2 \in \mathbb{X}, x_1 = x_2 : d(x_1, x_2) = 0$
2. **Positiviness:** $\forall x_1, x_2 \in \mathbb{X} : d(x_1, x_2) \geq 0$
3. **Simmetry:** $\forall x_1, x_2 \in \mathbb{X} : d(x_1, x_2) = d(x_2, x_1)$
4. **Triangular inequality:** $\forall x_1, x_2, x_3 \in \mathbb{X} : d(x_1, x_2) \leq d(x_1, x_3) + d(x_3, x_1)$

3.1.2 Metric Range and Nearest Neighbour Queries

A typical request is to find all the songs from $\mathbb{X}$ which have at most 15% of similarity with our query song q. This kind of queries are also known as *Metric Range Queries* on which the result is a subset $\mathbb{X}'$of $\mathbb{X}$ which satisfies $\forall x \in \mathbb{X}'; q, x_1, x_2 \in \mathbb{X} : d(x, d) \leq 0.15 \times max(d(x_1, x_2))$, where $0.15 \times max(d(x_1, x_2))$ represents the 15% of the maximum distance between elements in the space.

There is a good reason behind to choose the maximum distance between elements instead of the maximum distance allowed by our space, because the maximum distance is infinity, thus, the relative nature of the space does not allow such queries to be performed. In a strict sense, we can generalise metric range queries as follows:

$\forall x \in \mathbb{X}'; q, x_1, x_2 \in \mathbb{X} : d(x, d) \leq d$

Now, for our subset of waveforms if we want to obtain the 10 songs more similar to a certain one we are talking about a *Nearest Neighbour query*, on which the dataset is filtered and the result is a subset $\mathbb{X}'$ of $\mathbb{X}$ of fixed size k (in this case 10) satisfying $\forall x_1, x_2 ... x_n \in \mathbb{X}, x_1 < x_2 < .. < x_n; q \in \mathbb{X} : \mathbb{X}' = x_1, x_2, .., x_{10}$. Formally this can be defined as $\forall x_1, x_2 ... x_n \in \mathbb{X}, x_1 < x_2 < .. < x_n; q \in \mathbb{X} : \mathbb{X}' = x_1, x_2, .., x_k$.

3.1.3 Dimensionality crux

All the queries previously explanined requires an evaluation of every element on the index, thus any improvement on the running time can be only achieved by reducing the dataset to be compared, but this ignores the fact that we do not know how expensive it is to compute a distance between two elements. On high-dimensional spaces, distance computation can be a pain depending on the function d involved. This dependency on the space dimension is known as the *dimensionality crux*. There are many approaches to this problem such as dimensionaly reduction using techniques like *Self Organizing Maps* which work under the premise that the real dimensions of the problem are less than the ones given by the dataset, depends on a prior treatment of the data and requires the asumption of data loss.

So, dimensionality crux can be defined as the tradeoff between the amount of data available for computation (thus, the quality of our results) and the efficiency of the operations performed on this space.

3.1.4 Pivot and Permutant based indices

Given the dimensionality crux, we know that we need to reduce the dataset to compare in order speed up out queries, so, let's change the problem from searching similar songs to a geolocation problem, on which we need to find a point closer to another in the world map. If we were to find Curicó (Chile) without any prior knowledge of where it is, then the only option is to search on every label on the map in the hopes of finding the one that we desire to mark. This is known as *exhaustive search*, which is pretty much like performing a linear scan over the elements of $\mathbb{X}$.

Everything would be easier if we have an *index* to search on, a small database that can give us some insight on where the location belongs in order to reduce the problem size even if doesn't give us the exact position. We do not worry about dimensionality crux implcations at this point, since we are not quering the exact solution, rather a good aproximation of it. Now, assuming that we ask our index about the location of Curicó it can suggest how far it is from the major cities on the world (*pivot* cities), this kind of indices is known as *Pivot-based indices*, on which we don't store all the dimensions of the problem, rather than the distance of the element to certain *key elements* of $\mathbb{X}$. Then by applying the triangular inequality property of metric spaces we can triangulate a reduced area on the map in which we know that Curicó is situated.

So, assuming that we have a set $\mathbb{P} \subset \mathbb{X}$ of randomly chosen pivots, for a range query (q,r) on $\mathbb{U}$ we do not need to compute the distance of every element of $\mathbb{X}$ to our query object q since the triangle inequality follows that $\forall x \in \mathbb{X} : d(p_i,x) \leq d(p_i,q) + d(q,x)$ and also that $\forall x \in \mathbb{X} : d(p_i,q) \leq d(p_i,x) + d(p_i,q)$ thus we can approximate $d(q,x)$ for all elements on $\mathbb{X}$ only using the reference data given by the closeness of each element to their pivots.

This approach implies that $|\mathbb{P}|$ indices must be created in order to store the information needed for a pivot approach but since we can store them as sorted arrays of data we perform an approximate binary search on each one of the indices (which can be later retrieved using incremental sorting techniques [**?**]), thus the search complexity yields $|\mathbb{P}| \times log_2(|\mathbb{X}|)$ for the pivot discard stage plus linear time to retrieve all of the results from the resulting sets at the cost of of extra $|\mathbb{P}| \times |\mathbb{X}|$ space.

Taking into account how space expensive is to generate such indices, another approach is to search for Curicó not using the distances to the major cities at all and only using the information given by the closeness order to each one of the pivots generated in the earlier stage. In this case, such cities which acts as pivots are know as *permutants*. Then our approach now is to not store all the indices generated by the pivots, rather to map the entire dataset to another index which condenses this queries creating a single database in which the *real distance calculations* are replaced by computing the permutation distance between each one of the permutants.

This kind of indices are known as *permutation indices* and work under the premise that computing the permutation distance is much simplier than to compute the actual distance for every object in $\mathbb{X}$. Such distances can be computed by using *vector absolute distance* forumlas such as *Spearman's footrule* and *Kendall 's tau*.

3.1.5 Correlation coefficients and permutation distance metrics

One of the most well-known formulas for computing the correlation on two datasets is *Spearman's rho*, which gives an interdependence metric between two continous random variables. For our case, we expect to have two permutations which represents a ranking of each object (namely, our query object and a test object in the dataset) so, to measure the similarity between those two permutations is a feasible solution at first glance. But there are two problems given by both the nature of the data and it is implementation. First *Spearman's rho* formula is intended for continous random variables, as such we always have intermediate objects between each data in the dataset, which is not our case. Second, it is hard to compute continous variable on a computer, since we do not enjoy o abritrary precision on our current compute machines.

As such, we use the discrete version of correlation coefficients, which are also known as *absolute distance* between vectors. In our case we have the following formulas for each one of the before mentioned metrics:

Spearman's Footrule, which measures the total element displacement between the identity permutation:

$$F(\sigma) = \sum_i (i - \sigma(i))$$

Kendall's tau, which measures the total number of pairwise inversions:

$$K(\sigma) = \sum_{i,j:i>j} (\sigma(i) < \sigma(j))$$

3.2 Field Programmable Gate Arrays

3.2.1 Overview

Field Programmable Gate Arrays (or FPGA for short) are programmable devices that contains logical blocks arranged as a grid that can be configured and interconnected as will in a *in-situ* way. The programable feature of this chips enables the replication of simple logic circuits sucj as logical gates to complex designs such as *Digital Signal Processors*. Commonly these chips are used in *Application-Specific Integrated Circuits* prototyping because the process behind its programming is similar to *waffer synthesis* used on the creation of ASICs and their design flow flexibility, but their energy consumption, thermal disipation and performance are worse than a dedicated circuit. The internal resources of an FPGA chip consist of a matrix of configurable logic blocks (CLBs) surrounded by a periphery of I/O blocks. Signals are routed within the FPGA matrix by programmable interconnect switches and wire routes. Because of the latter point, those chips are commonly used on robotics and hardware development.

3.2.2 FPGA Programming and its difficulties

Most programming on FPGA is done using *hardware description languages* such as *VHDL* and *VERILOG* which describes the circuit to be loaded onto the FPGA using a *register-transfer level* abstraction of synchronous digital circuit models a in terms of the flow of digital signals between each one of the hardware registeres and the logical operations performed on those signals as well. This generates a high-level representation of a circuit, from which lower-level representations and ultimately actual wiring can be derived.

As such, a single FPGA can replace thousands of discrete components by incorporating millions of logic gates in a single integrated circuit (IC) chip. Given that FPGAs are huge fields of programable gates, its programming allows the creation of multiple hardware paths, delivering a trully parallel nature in which different processing operations do not compete each other for resources (as opposite in modern CPUs in which all programs running on the OS are competting for the available resources). At the same time, hardware execution of the problem provide more performance than most processor-based solutions as well as a higher throughput of data, which becomes the main goal in FPGA programming.

FPGAs give software developers a promise of a greater performance than processor-based solutions, but it comes of a cost that all operations must be implemented as it were directly on hardware, thus, for most computer engineers there is a wide knowledge gap when it comes to use it for actual data processing.

3.2.3 High-level synthesis

In order to turn the hardware description into code, a process known as *logical synthesis* is performed, which takes an abstraction of the circuit behavour as RTL and then turns it into a logic implementation in terms of logic gates and blocks. Then, a *bitstream* is generated which is transfered to the FPGA to reconfigure its grid.

3.2.4 Interconnection between Programable Logic and Processing Systems

3.3 Adapteva Parallela

3.3.1 Overview

The Adapteva Parallella is a The Parallella platform is an open source, energy efficient, high performance, credit-card sized computer based on the Epiphany multicore chips developed by Adapteva in 2012 originated as a Kickstarter project marketed as “A Supercomputer for everyone”. The goal behind its creation is to create a low-power small heterogeneus architecture for supercomputing experimentation.

3.3.2 Hardware revisions

Parallella board currently is sold in three major versions, namely *Parallella-16 Micro Server*, *Parallella-16 Desktop Computer* and *Parallella-16 Embedded Platform* as shown

	Parallella-16 Micro Server	**Parallella-16 Desktop Computer**	**Parallella-16 Embedded Platform**
Use Case	Ethernet connected headless server	A personal computer	Leading edge embedded systems
Processor	Dual-core 32-bit ARM Cortex-A9 with NEON at 1 GHz (part of Zynq Z7010 chip by Xilinx)	Dual-core 32-bit ARM Cortex-A9 with NEON at 1 GHz (part of Zynq Z7010 chip by Xilinx)	Dual-core 32-bit ARM Cortex-A9 with NEON at 1 GHz (part of Zynq Z7020 chip by Xilinx)
Coprocessor	16-core Epiphany III multi-core accelerator (E16)	16-core Epiphany III multi-core accelerator (E16)	16-core Epiphany III multi-core accelerator (E16)
Memory	1 GB DDR3L RAM	1 GB DDR3L RAM	1 GB DDR3L RAM
Ethernet	10/100/1000	10/100/1000	10/100/1000
USB	N/A	2× USB 2.0 (USB 2.0 HS and USB OTG)	2× USB 2.0 (USB 2.0 HS and USB OTG)
Display	N/A	Micro-HDMI	Micro-HDMI
Storage	16 GB microSD, up to 32GB	16 GB microSD, up to 32GB	16 GB microSD, up to 32GB
Expansion	N/A	2 eLinks + 24 GPIO	2 eLinks + 24 GPIO
FPGA Available Logic Cells	28K programmable logic cells	28K programmable logic cells	80K programmable logic cells
FPGA Available DSP Slices	80 programmable DSP slices	80 programmable DSP slices	220 programmable DSP slices
Weight	36 g (1.3 oz)	38 g (1.3 oz)	38 g (1.3 oz)
Size	3.5 mm × 2.1 mm × 0.625 mm (0.1378 in × 0.0827 in × 0.0246 in)	3.5 mm × 2.1 mm × 0.625 mm (0.1378 in × 0.0827 in × 0.0246 in)	3.5 mm × 2.1 mm × 0.625 mm (0.1378 in × 0.0827 in × 0.0246 in)
SKU	P1600-DK-xx	P1601-DK-xx	P1602-DK-xx
HTS Code	8471.41.0150	8471.41.0150	8471.41.0150
Power	USB-powered (2.5 W) or 5 V DC (∼5 W)	USB-powered (2.5 W) or 5 V DC (∼5 W)	USB-powered (2.5 W) or 5 V DC (∼5 W)

Table 3.1: Comparison between different Adapteva Parallella Boards

in Table 2.1. The main differences between each one of the boards is the available ports and the FPGA chip used on each one of them.

3.3.3 Hardware Architecture

The high level overview of the Parallella hardware system is shown in the Figure 2.2. Here is possible to identify the core components of the board, namely the Epiphany16 co-processor and the Zynq SoC on which the Parallella board is based on. The board enables multiple configurations for interconnection between other devices as well as other Parallella boards itself which are detailed in Figure 2.3.

For the system embedbed on the parallella board to be functional and have a basic communication channel between the ARM-A9v7 and the Epiphany chip, only a subset of FPGA blocks (framed in red in the picture) is needed. These blocks are:

- **AXI-MASTER:** A master port on the AXI bus for communication between DRAM and the Epiphany.
- **AXI-SLAVE:** A slave port on the AXI bus used to access Epiphany and FPGA resources
- **eLink** The link-port interface to the Epiphany chip.
- **"Glue-Logic"** This logic implements the interface between AXI ports and Epiphany link port. The system level registers are implemented in this logic too.

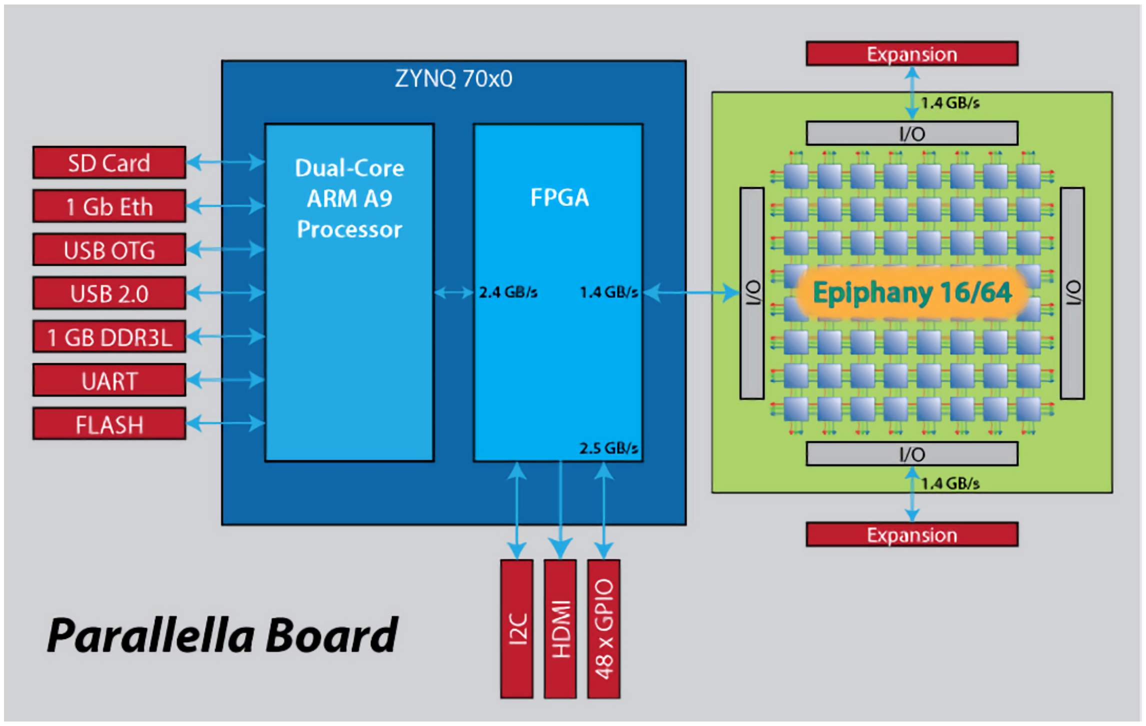

Figure 3.1: Adapteva Parallella Architecture Overview

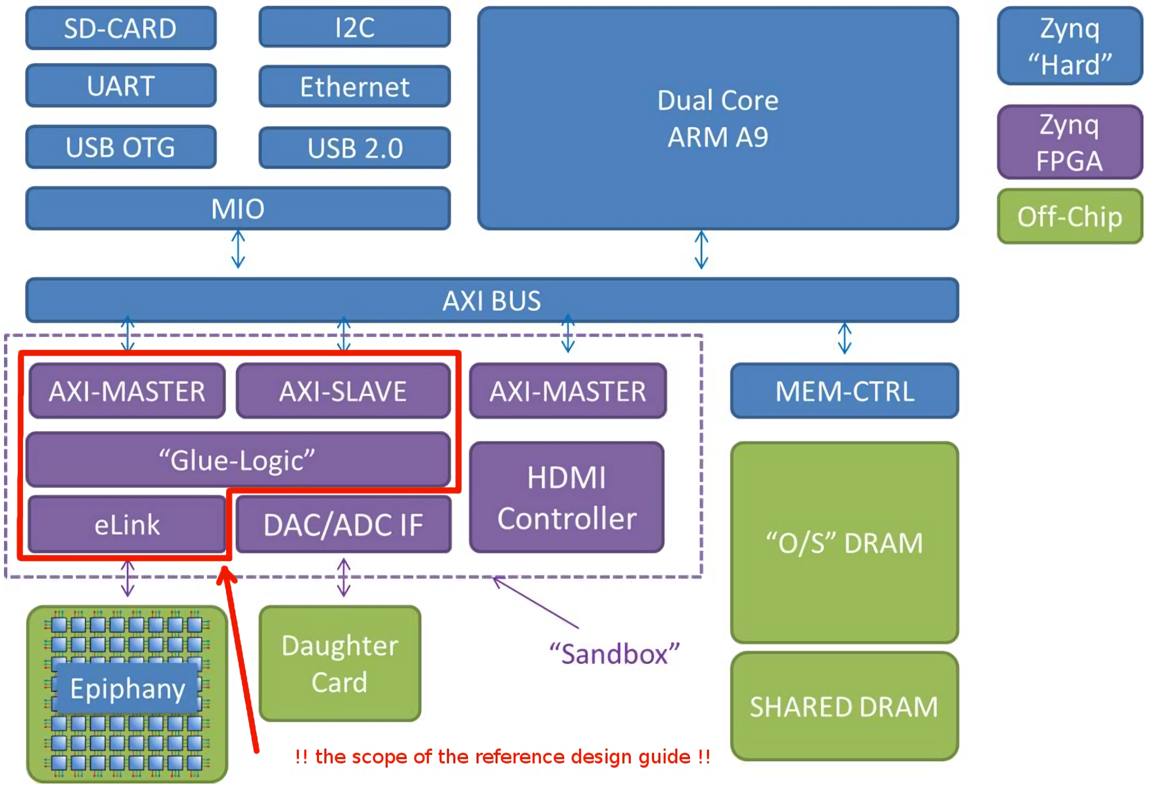

Figure 3.2: Adapteva Parallella System Architecture. Physical resources being highlighted in blue, logical resources being highlighted in purple and off-chip resources being highlighted in green.

3.4 Hardware acceleration

3.4.1 GPGPU

3.4.2 ASIC

3.4.3 FPGA

3.4.4 Design synthesis

3.4.5 High Level Synthesis

3.5 Embebbed Linux

3.5.1 Linux kernel

3.5.2 Modules

3.5.3 Devicetree

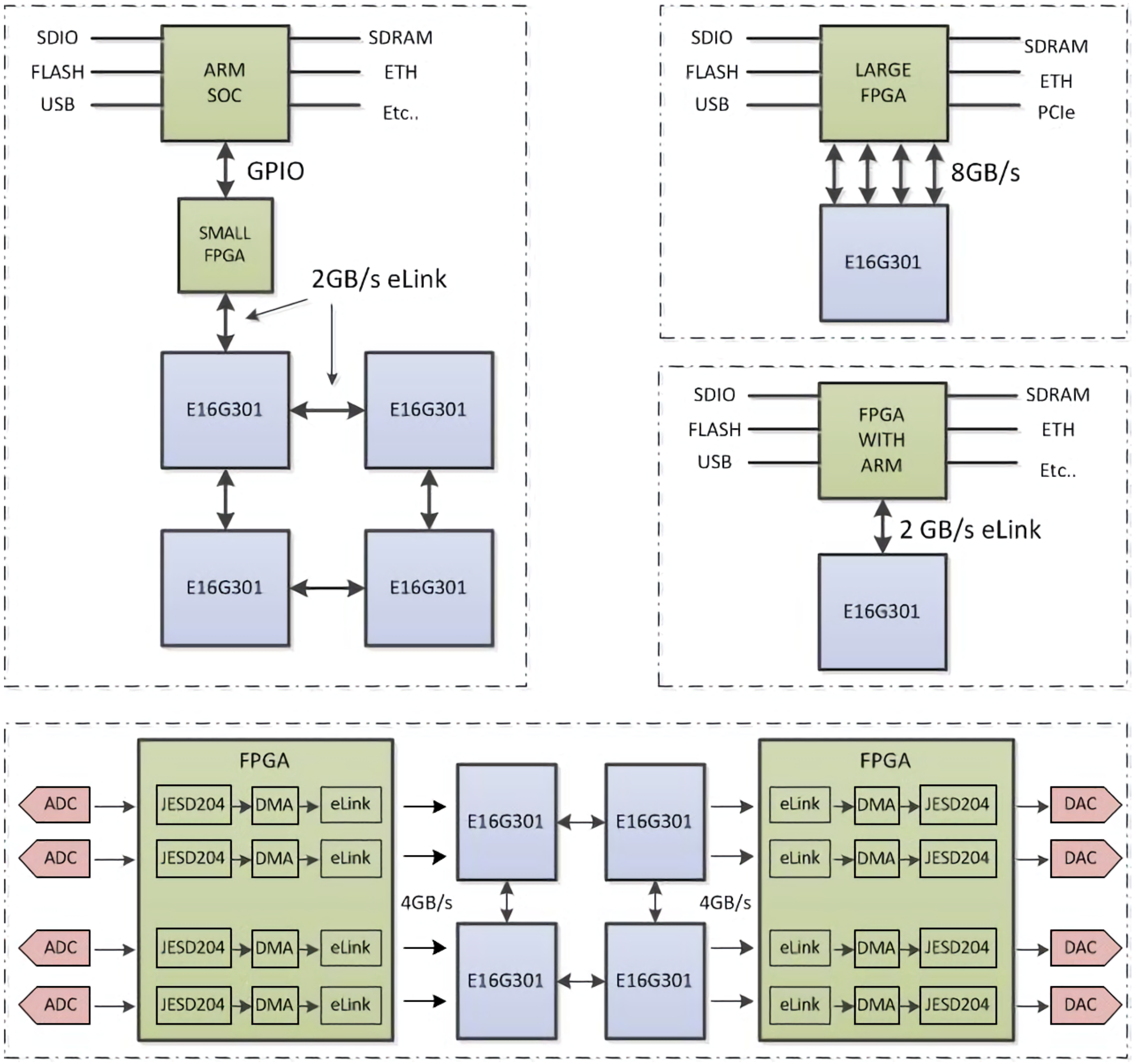

Figure 3.3: Adapteva Parallella possible hardware configurations

4. Metric Space indexing

4.1 Dataset description

4.2 Implemented algorithm

5. Software implementation analysis

5.1 Algorithm analysis

5.1.1 Index generation

5.1.2 Approximate search

5.2 Code analysis and benchmarking

5.2.1 Permutation distance

5.2.2 Permutation generation

6. Accelerator Implementation

6.1 High Level Synthesis

6.1.1 Overview

6.1.2 Latency

6.1.3 Thoughput

6.1.4 Directives

6.1.5 Impact of coding style

6.2 Permutation distance

6.2.1 Analysis

6.2.2 Implementations

6.3 Permutation generation

6.3.1 Analysis

6.3.2 Implementations

7. Hardware-Software interoperation

7.1 AXI4 Protocol

7.1.1 AXI4 Protocol

7.1.2 AXI4Lite

7.1.3 AXI4Full

7.1.4 AXI4Stream

7.2 Direct Memory Access

7.2.1 AMBA & Devicetree

7.2.2 Modules and device drivers

7.3 Implementation

8. Results

8.1 Original implementation benchmarks

8.2 Accelerated implementation benchmarks

8.3 Comparison between results

9. Conclusions

Glossary

El primer término: Este es el significado del primer término, realmente no se bien lo que significa pero podría haberlo averiguado si hubiese tenido un poco mas de tiempo.

El segundo término: Este si se lo que significa pero me da lata escribirlo...

ANNEX

A. HLS IP C++ code for Even-Odd sorting network example

```
#include <hls_stream.h>
#include <ap_axi_sdata.h>
typedef ap_axis <32,2,5,6> intSdCh;
#define INTS_TO_PROCESS 10000
void doGain(hls::stream<intSdCh> &inStream, hls::stream<intSdCh> &outStream){
#pragma HLS INTERFACE axis port=outStream
#pragma HLS INTERFACE axis port=inStream
    intSdCh valIn[INTS_TO_PROCESS];
    int idx;
    for (idx = 0; idx < (INTS_TO_PROCESS); idx++){
    #pragma HLS PIPELINE
        valIn[idx] = inStream.read();
        intSdCh valOut;
    }
    for (int stage = 0, temp, i  ; stage < INTS_TO_PROCESS ; stage++){
    #pragma HLS UNROLL factor=2
        if (stage & 1){
            even: for(i=2; i<INTS_TO_PROCESS; i+=2){
            #pragma HLS PIPELINE
            #pragma HLS UNROLL factor=10
                if (valIn[i].data < valIn[i-1].data){
                    temp = valIn[i].data;
                    valIn[i].data = valIn[i-1].data;
                    valIn[i-1].data = temp;
                }
            }
        }
        else{
            odd: for(i=1; i<INTS_TO_PROCESS; i+=2){
            #pragma HLS PIPELINE
            #pragma HLS UNROLL factor=10
                if (valIn[i].data < valIn[i-1].data){
                    temp = valIn[i].data;
                    valIn[i].data = valIn[i-1].data;
                    valIn[i-1].data = temp;
                }
            }
        }
    }
    for (int idx = 0; idx < (INTS_TO_PROCESS); idx++){
    #pragma HLS PIPELINE
        outStream.write(valIn[idx]);
    }
}
```