Optimizing Multicore System Performance Using Dynamically Reconfigurable Architectures on FPGAs

Prashanth Guledal Lakshamana

CE-MS-2018-17

Delft University of Technology

Abstract

Processor architecture is continuously evolving. As the trend predicted by Moore's law is nearing its end, the focus of designing processors has shifted from high-frequency single-core systems to the medium frequency multicore system to a relatively lower frequency many-cores system, in the hope of extracting more performance while keeping power consumption in check. To satisfy a spectrum of applications, modern processors employ central processing units (CPUs) for serving a wide variety of general-purpose applications, while general purpose - graphics processor units (GP-GPUs) are used for highly parallel applications. This thesis provides an alternative, called dynamic platform, by switching between a sequential processor for serving sequential applications and parallel processor for serving parallel applications on a Zynq FPGA (Field Programmable Gate Array). The first part of this thesis analyses and designs the model with suitable simulations to know the trade-offs. From the model, it is clear that towards the extreme ends of the application spectrum where either high level of parallelism exist or high level of sequential operations exist, GPU and CPU respectively outperform the dynamic platform. However, there exists a region suitable for the dynamic platform where the applications are neither too parallel nor highly sequential. To implement the model on FPGAs, suitable open-source softcores are researched and selected. ρ-VEX dual-core and Microblaze softcores are implemented for catering to sequential applications, and ρ-VEX many-core softcore is implemented for serving parallel applications. These softcores are evaluated against three benchmarks– Image processing (parallel), CRC (sequential) and Hash (sequential). Finally, the dynamic platform analysis is done, and the results prove that on average the performance on the dynamic platform is better than considering either the sequential (ρ-VEX dual-core) or parallel (ρ-VEX manycore) platform alone. The speedup of the dynamic platform ranges from 1.45 to 2.9 (average: 1.61) with respect to sequential platform and from 1.02 to 1.60 (average: 1.44) with respect to parallel platform. In the current state of FPGA technology, the dynamic platform does not perform better than CPU and GPU on average for the considered benchmarks. The result is a fully functional open-source dynamic platform, which can switch between two (or three) architectures at run-time, depending on the application characteristic (sequential or parallel).

TU Delft Delft University of Technology

Quantum & Computer Engineering

Optimizing Multicore System Performance Using Dynamically Reconfigurable Architectures on FPGAs

by

Prashanth G L

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Engineering

at the Delft University of Technology,
to be defended publicly on Monday August 20, 2018 at 01:30 PM.

Supervisor:	Dr.ir. Zaid Al-Ars	
Thesis committee:	Dr.ir. Zaid Al-Ars,	TU Delft
	Dr.ir. J.S.S.M. Wong,	TU Delft
	Dr.ir. T.G.R.M. van Leuken,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Dedicated to my family and friends

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

CPU Central Processing Unit

GPU Graphics Processing Unit

GP-GPU General Purpose GPU

CUDA Compute Unified Device Architecture

FPGA Field Programmable Gate Array

PAL Programmable Array Logic

PLA Programmable Logic Array

PLD Programmable Logic Devices

CPLD Complex PLD

DSP Digital Signal Processor

CLBs Configurable Logic Blocks

LUT lookup table

LEs Logical Elements

SOC System-On-Chip

NoC Network-On-Chip

RC Reconfigurable Computing

FIT Failures In Time

MTBF Mean Time Between Failures

PS Processing System

PL Programmable Logic

APU Application Processing Unit

MPE Media Processing Engine

FPU Floating Point Unit

MMU Memory Management Unit

OCM On-Chip Memory

SCU Snoop Control Unit

ICT Information and Communication Technology

HD High Definition

LPSoCD Low Power System on Chip Design

MTCMOS Multi-Threshold CMOS

PG Power Gating

RPG Power Gating with Retention

MSV Multiple Supply Voltages

DVS Dynamic Voltage Scaling

ABB Active Body Bias

AVS Adaptive Voltage Scaling

OoO Out of Order

NRE Non-Recurring Engineering

SMs Streaming Multiprocessors

SFUs Special Functional Units

SB Switch Boxes

CB Connection Boxes

RISC Reduced Instruction Set Architecture

ARM Advanced RISC Machine

ISA Instruction Set Architecture

AI Artificial Intelligence

IOT Internet Of Things

APB Advanced Peripheral Bus

IPC Instructions Per Cycle

LMB Local Memory Bus

BRAM Block RAM

RF Reduction Factor

VLIW Very Long Instruction Word

FPU Floating Point Unit

APIs Application Programming Interfaces

OpTiMSoC Open Tiled Manycore System-on-Chip

MCAPI Multicore Communications API

MTAPI Multicore Task Management Working Group API

NPU Network Processing Unit

MPI Message Passing Interface

DMA Direct Memory Access

L1 Level 1

L2 Level 2

PCAP Processor Configuration Access Port

DevC Device Configuration

ICAP Internal Configuration Access Port

UART Universal Asynchronous Receiver-Transmitter

LMB Local Memory Bus

GCC GNU Compiler Collection

ACP Accelerated Coherency Port

DU Delay Unit

CRC Cyclic Redundancy Check

MP Mega Pixels

MC Million Characters

ACKNOWLEDGEMENTS

I would like to thank Zaid Al-Ars for giving me an opportunity to work on this new and exciting project, and being my wonderful advisor.

I would like to acknowledge Saevar and Jacko for their assistance with the ρ-VEX processor.

I would like to express by gratitude to Joost and Jeroen for guiding me in the initial stages of my thesis.

I am really grateful for my parents for their support in inspiring me at difficult times.

Prashanth G L
Delft, The Netherlands
August 2018

1

Introduction

Science is increasingly answering questions that used to be the province of religion.
- Stephen Hawking

This thesis documents the modeling and design of a dynamically reconfigurable processor architecture. This chapter aims to give a concrete introduction to the thesis topic. First, the context of the thesis is explained in Section 1.1. This is followed by the discussion of the motivation for developing the reconfigurable architecture in Section 1.2. Subsequently, problem statement, methodology and goals are elaborated in Section 1.3. Finally, the organization of this thesis document is reported in Section 1.4.

1.1. Context

The industrial and scientific community have invested a great amount of time in developing and exploring reconfigurable architectures since the 1980s. Moore's law encourages developers to reassess their approaches and forces them to approach new ideas to keep up with the challenges of the current technology, as long as the law is valid.

Amdahl's law in the multi-core era [1], which is a highly influential paper in the last decade or so, delves into the different kinds of architectures with varying performances. This paper is highly relevant for the scientific community of multicore developers as it connects Amdahl's law to the available multicore hardware. It mentions mainly three models which cover a large portion of the processor architectures in use. These models are discussed next and are the backbone of this thesis.

First, the symmetric model is discussed in which all the cores have the same cost. This implies that all the cores give the same level of performance for a given application under consideration. Further, this model can be of two types. One is the architecture wherein performance is derived from a small number of large cores. For example, the Central Processing Unit (CPU) comes under this category. Here the cores are big and complex containing caches of large sizes, long pipelines, sophisticated super-scalar hardware such as branch prediction, out of order execution, etc., and give excellent thread level performance. They are mainly suited to desktop applications and in particular sequential applications. Second subtype here is the one which consists of a huge number of small cores. For example, the Graphics Processor Unit (GPU) belongs to this category. Each core is very small and minimal. It is designed to give high throughput for parallel or high-performance computing applications. It hides the latency of computation and memory access by executing 1000s of threads simultaneously.

Second main model is the asymmetric model which contain one or more powerful cores and many small cores. In particular, this heterogeneous model comprises of one large core with enormous computing resources to efficiently execute sequential applications, while the parallel applications are dealt with the remaining small cores. This seems to suggest that this model covers both bases to execute sequential and parallel programs by utilizing suitable cores. However, one big disadvantage of this model is the idle time. Suppose a serial application is running on the big core, then all the small cores are idle which results in inefficient use of the underlying hardware and vice versa.

Third and final model is called the dynamic multicore model, which is the base on which this thesis is built. This model presents an ideal solution to use the hardware to the fullest extent. The main idea here is to switch the architecture depending on the application under consideration using the many small cores available. Suppose the application to be executed is highly sequential, then it is handled by the large cores that are created by combining many small cores using thread-level speculation or helper threads. If the application is highly parallel, then the performance is extracted from all the small cores existing. On paper, this seems to be the best performing architecture for any type of application. But there is no practical implementation of this model yet. This is the main reason we consider this dynamic architecture to implement in this thesis, analyze and remodel further for the realistic performance indications.

1.2. MOTIVATION

Changing the architecture of the processor depending on the level of parallelism available in the application under consideration is a powerful concept. Indeed, it makes us think that this might be the architecture which can provide good alternative to the traditional desktop (CPU+GPU) computers. This implies that the dynamic nature of our reconfigurable processor could make it suitable for a huge variety of applications such as server market, cloud computing applications, etc.

The alternative is to prove that this model is incorrect in assuming high performance for the dynamic architecture, and identify the limitations of the model so that it can predict the realistic performance indication. Either way, it will lead to clear answers on the feasibility of employing this architecture in real-world applications.

1.3. PROBLEM STATEMENT AND METHODOLOGY

Reconfigurable architecture is not a new concept, but either it is used for a specific block as a partial reconfiguration or changes it entirely over time so that the new configuration provides improved performance. In this regard, there doesn't exist any intelligent architecture which can change itself completely depending on the underlying architecture. So, the driving question here is that if our dynamic architecture provides the best performance, then why doesn't it exists yet? In this thesis, we model and design this architecture on Field Programmable Gate Arrays (FPGAs) to clearly understand its behavior and performance implications.

The main problem statement of this thesis is:

> On average, for the sequential and parallel applications, can the performance on our designed architecture be better than on the CPU, the GPU or the static softcore?

The sophisticated approach needs to be followed in order to answer this question. First some simple kernels need to be simulated on CPU, GPU and use extrapolation to estimate the performance on our ideal architecture. This is followed by the selection of suitable softcores in order to cater to both sequential and parallel needs of the program. We chose ρ-VEX manycore as the parallel softcore and ρ-VEX dual-core as well as Microblaze single-core for the sequential purpose. Subsequently, both the softcores are synthesized on FPGA and detailed performance measurements are done to analyze the trade-offs.
The main goals of this thesis are summarized as follows:

1. Model the performance for simple kernels by simulating them on CPU, and GPU.
2. Determine the region of interest in which the dynamic platform performs well and identify the feasibility of implementing such a dynamic architecture.
3. Designing and implementing a platform capable of switching between two architectures:
    - Parallel architecture to deal with the parallel applications.
    - Sequential architecture to deal with the sequential applications.
4. Performance analysis of the designed platform along with the discussion on the important trade-offs.

1.4. ORGANIZATION

The remainder of this thesis is split into five chapters, and they are organized as follows:

- Chapter 2. **Background** In this chapter, significance and challenges of reconfigurable computing are elaborated. Then, we take a closer look at the processing architectures (CPU, GPU and FPGA) that are used in this thesis and their unique characteristics. Further, selection of softcores, required to implement a dynamic platform, is explored in detail. Two types of open-source softcores are researched and selected. One corresponds to single-core/dual-core softcore and the other corresponds to scalable manycore softcores.

- Chapter 3. **Conceptual Model** This chapter delves into conceptual modeling of the dynamic platform. It begins with a brief motivation which led to the development of this concept. Then, the model is introduced with a description of processing frameworks and two kernels, that are executed on these frameworks. Subsequently, modeling is carried out with real simulated values on CPU and GPU. But due to the limitation faced in this model, the performance values are then suitably assumed to create a powerful abstraction model which is simple and intuitive to understand. Finally, the model is successfully completed by injecting a realistic scenario so that meaningful insights are obtained.

- Chapter 4. **Implementation** This chapter focuses on the implementation of the processing architectures. First, the ρ-VEX manycore processor, consisting of 10 cores, is implemented. Then, the Microblaze single-core processor is implemented in a high-performance configuration. Subsequently, the ρ-VEX dual-core processor, consisting of two level caches that can fill an entire FPGA area, is implemented. Further, the design of software for the dynamic platform is discussed.

- Chapter 5. **Experiments, Results and Analysis** In this chapter, detailed discussion is done on the experiments and results for the implemented architectures. First, the benchmarks used in the experimentation are described. Then the evaluation of the implemented platforms is carried out against the described benchmarks. For the ρ-VEX dual-core platform, a special analysis is done to understand the effect of varying cache size and the distance of data-path from DDR memory to Level 2 (L2) cache. Further, the dynamic platform analysis, which is a core part of this thesis, is elaborated along with the implications of switching overhead in terms of time. Furthermore, the comparison of the ρ-VEX softcore with an on-chip hard processor is discussed.

- Chapter 6. **Conclusion and Future Work** In this chapter, first, a brief summary of the overall work is elaborated. Then, the main contributions of this thesis are mentioned. Further, the future work is discussed which can add value in the direction of this thesis.

b

2

Background

The whole of science is nothing more than a refinement of everyday thinking.
- Albert Einstein

This chapter serves to introduce preliminaries concerning Reconfigurable Computing (RC). First, on account of this thesis based on the dynamic reconfigurability, a detailed look on why RC plays a promising role in the future of computing is given in Section 2.1. This is closely followed by the current challenges in the area of RC in Section 2.2. Subsequently, Section 2.3 elaborates on the traditional computing platforms to see their unique characteristics. Furthermore, a brief survey on the existing softcores is done in Section 2.4. In the same section, discussions are made to select a suitable softcore for sequential as well as parallel processor architectures.

2.1. Significance of Reconfigurable Computing

It is very important to understand why we need to envision RC as the future prospect of computing. This section details the factors contributing to the growth in interest and opportunities in the area of RC, and lays emphasis on moving RC from a specialized area to mainstream computing. The dominance of basic computing paradigm is starting to loosen its intensity due to the emergence of RC applications - reflecting major changes in the adoption of high-performance scientific computing methods and common embedded systems, as well as bringing a new promise of affordable computing [2]. This, in turn, makes the desktop supercomputing near.

2.1.1. Energy Consumption Trends of Computing

Electricity consumption of the computer-based infrastructures all around the world has increased drastically in the past two decades. The contribution of Information and Communication Technology (ICT) in electricity consumption was around 8% in 2008 and is predicted to rise more than 14% (i.e. 1/7th) in 2020 [3]. One other research [4] indicates that if the current trends continue, the growth by a factor of 30 in the consumption of electricity by ICT is estimated to take place by the year 2030. Main contributing factors to this growth include ever-expanding wireless internet and its user base, preference towards more video on demand, High Definition (HD) and 4K videos, developing trend of rising electronic book usage, cloud computing capturing increasing share of high performance computing and server market, etc.

Computers are a basic necessity for the millennial generation. They are indispensable for a wide variety of reasons [5], which mainly includes running millions of lines of code for various software. However, the increased power consumption threatens the survival of existing cyber systems. The power bill is now a major concern not just for top companies like Google, Amazon, Facebook and Microsoft. It is predicted that in the near future electricity bill of most computing tasks will be

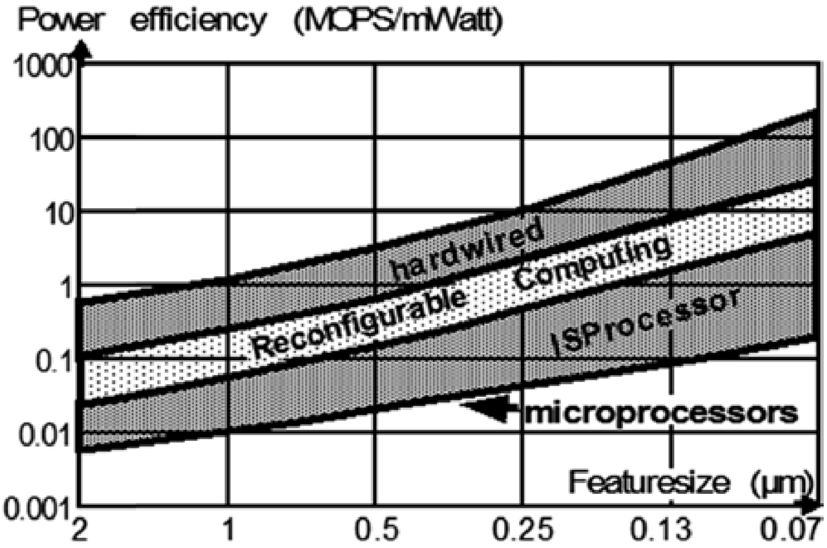


Figure 2.1: Figure showing the better power effciency by accelerators [8].

remarkably higher than the cost of the equipment itself [6]. This is evident by the fact that the cost of data center is already calculated by the monthly power bill and not by the maintenance or hardware costs [7].

2.1.2. GREEN COMPUTING: IS IT ENOUGH?

There are many conservative methods being used in the area of green computing in the hope of saving energy by designing efficient power modules. For instance, LED flat panel displays are used over LCD-based displays as LED displays consume less power. The potential to save power is less than an order of magnitude: maybe, a factor of about 3–5 [2].

A separate but closely related to this concept is the area of Low Power Circuit Design, also called Low Power System on Chip Design (LPSoCD). Over the last few decades, several well known methods were developed in this area, such as: Multi-Threshold CMOS (MTCMOS), Power Gating (PG), Power Gating with Retention (RPG), Multiple Supply Voltages (MSV), Dynamic Voltage Scaling (DVS), Active Body Bias (ABB), Adaptive Voltage Scaling (AVS), etc. The order of magnitude of the benefit to be expected here is very low, for example, by MSV in using 3 Vdds the power reduction ratio at best is about 0.4 [7]. One important point to note here is that LPSoCD is mainly applied to ASIC or hard-wired designs. Only 3% of all design starts are ASIC designs [2] with a trend leading further down. The fact of the matter is that the low power design principles are used for creating power-efficient FPGAs. However, “Green Computing has become an industry-wide issue: incremental improvements are on track [9]" and “we may ultimately need revolutionary new solutions [10]". Further, “we will ultimately also need revolutionary solutions (like RC), since we need much higher efficiency [2]".

2.1.3. RC IMPLICATIONS ON ENERGY CONSUMPTION

The idea of saving energy using RC is not a new one. Being very important to massively reduce the energy consumption of computing, by up to several orders of magnitude, RC is extremely important for the survival of the world economy [2]. It has been reported in [8] more than a decade ago, that for a given feature size, microprocessors using traditional compilers have been up to 500 times more power hungry than a pure hardware mapping of an algorithm in silicon as it can be seen in Fig. 2.1. Fig. 2.2 shows a few speedup factors picked from literature (ref in [11]). The literature also provides information that about 4 orders of magnitude speedups can be obtained from migrating

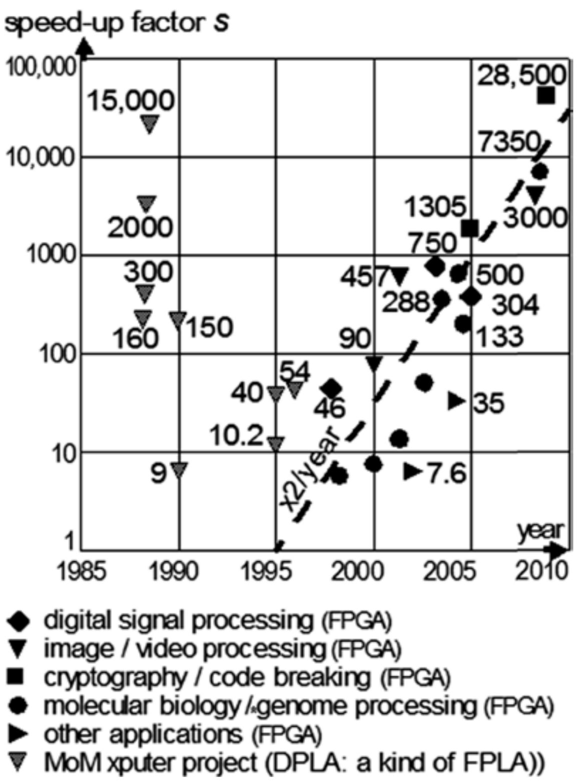


Figure 2.2: Various Speed-Up factors achieved in wide areas of applications [2].

the software to FPGA.

Energy savings are reported in [12] and is shown in Table 2.1. The table shows that energy savings are roughly one order of magnitude less than the speed-up. The influence of FPGA is not confined just to embedded systems but can be applied to all areas of scientific computing, which demands high performance.

2.1.4. HOW RC CAN WIN OVER TRADITIONAL COMPUTING

Traditional computing mainly consists of Von-Neumann architectures. Von-Neumann machines, in a simplest way, consist of a processor (CPU), a memory and a datapath connecting these two units. Although this architecture dominates the computing market, it has been criticized widely and regularly for various reasons. Two main reasons are mentioned here. One, it is based hugely on instruction streams that are very memory-cycle-hungry, and the other is an architectural problem, the memory wall: the access time to RAM outside the processor chip is slower by a factor of about 1,000, than to on-chip memory while the difference is growing by 50% every year [13]. This problem is further made worse by the fact that a big portion of the traffic through the channel, connecting the processor and memory, is not useful data, but merely names of data, as well as operations and data used only to compute such names which means that the back-and-forth movement of data

Table 2.1: Energy savings and speed up factors using FPGAs [12]

		Save Factor		
Applications	Speed-up factor	Power	Cost	Size
DNA and protein sequencing	8,723	779	22	253
DES braking	28,514	3,439	96	1,116

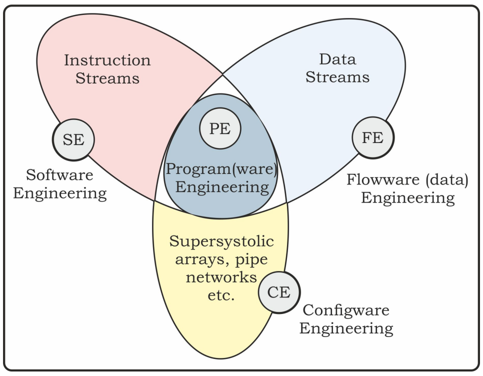

Instruction Streams
Data Streams
SE
PE
FE
Software Engineering
Program(ware) Engineering
Flowware (data) Engineering
Supersystolic arrays, pipe networks etc.
CE
Configware Engineering

Figure 2.3: Copernican model of computing

only makes the operation slow [14].

This kind of traditional CPU-centric world model is becoming obsolete. The main hindrance here is that as it is based only on instruction streams, it hides other important perspectives such as structural and data stream aspects. The author in [2] proposes a hetero model, called copernican model, shown in Fig. 2.3, which, in addition to instruction streams (software engineering), also includes structures (configware engineering) and data (flowware engineering) streams, and supports time to space mapping, since scaling laws favor reconfigurable spatial computing over temporal computing. The author further argues that this type of model leads to the better development of computing systems to reach new horizons of research and development in RC as well as computer science.

2.2. CHALLENGES OF RECONFIGURABLE COMPUTING

The previous section showed the promising prospects of RC. This section details the current challenges being faced in the field of RC. The widely-known fact about RC is that the reconfigurable hardware takes approximately three orders of magnitude higher Area*Time*Power product compared to ASICs [15]. Nevertheless, the RC can offer better power/performance ratio and faster execution times than generic machines. This can be easily justified, as at a given point in time only a small fraction of resources of the general purpose processors are available for useful computations, the remaining resources constitute for a huge amount of memory in the form of the memory hierarchy and other complicated entities such as branch predictor, Out of Order (OoO) execution, etc., which only indirectly enhance the performance.

RC acts as a promising alternative to offer solutions for many severe problems put forward by the current semiconductor technology trends such as performance, power, and reliability. The area overhead of the reconfigurable substrate is turning out to be less vital because of the power density and increasing number of transistor count per fixed area. The area overhead can be a blessing in disguise, as on-demand reconfigurability serves to deliver excellent solutions for fault-tolerance by isolating and correcting defective blocks [16]. However, several important issues need to be addressed before RC can become a mainstream computing solution. These issues are briefly discussed further.

2.2.1. RUN-TIME RECONFIGURATION DELAY

An important part of the reconfigurable architectures is to provide the ability to adapt to the environment or the various needs of the application under consideration. This means that software is not the only changing factor but also the underlying hardware. Reconfiguration is useful in a number of ways such as adding or removing specific functional modules and accommodating to the dynamic needs of the application. One of the main challenges being faced in the area of RC is the reconfiguration delay. For example, any real-time high-performance application is sensitive to the reconfiguration speed. Two main reasons for the reconfiguration delay stem from the bottlenecks involved in configuration interfaces and reconfiguration issues in the system design methodology itself [2]. Therefore, there is a need to improve runtime support in order to overcome this issue.

2.2.2. Reliability

Reconfigurable Platforms play a very important role in the field of fault-tolerant computing. Their inherent flexibility adds to the fact that any faulty components are easily isolated, corrected and/ or replaced. The challenge here is to develop good techniques to support the reconfigurable platforms with the run-time support that can recover from transient faults. Even though the inherent advantage of reconfigurable hardware is redundancy, we still lack however efficient methods to exploit this characteristic and provide a fault tolerant reconfigurable platform [2].

2.2.3. Tool Support

Efficient tool support is essential for the development of any FPGA hardware. They are fundamental to the designer's productivity and quality to develop complex and large reconfigurable systems. As the FPGAs tend to increase its resources over time, the design time increases gradually as the tools tend to explore the huge design space in order to meet the underlined constraints such as power, timing or efficiency. The current challenge in developing tools that target RC systems is to create a coherent compilation tool-chain that covers all the main steps of synthesis and analysis, including capturing domain-specific knowledge, profiling, design space exploration, multi-core partitioning, system partitioning, data representation optimization, static and dynamic reconfiguration, optimal custom instruction set generation, and functional simulation [16].

Run-time support is another important aspect of the reconfigurable system. Adjusting to the underlying hardware on demand in terms of power, efficiency or other application needs is achieved better at run-time than at the compile time. In the current state of research, we know how to develop a runtime system for a fixed hardware platform, however building a runtime system for dynamically changing hardware platforms remains an open issue [2].

2.2.4. Memory Architecture Limitations

The efficiency of the reconfigurable system is influenced heavily by the memory architecture built around the reconfigurable system as well as the memory structure of the reconfigurable platform itself. To create a suitable memory architecture which can provide enough flexibility (number of banks, width and depth) to a general-purpose reconfigurable platform running a multitude of applications is a challenging issue. Furthermore, matching the memory bandwidth with the computation power of the system can improve system's performance [16]. There is some research done already in this area, but it has not progressed further in that direction. The authors in [17] developed Flux Caches, a mechanism in which the cache organization and implementation is dynamically changed to adjust the memory hierarchy of a system on demand based on the application needs.

2.2.5. Computational Power Efficiency

Owing to low non-recurring engineering (NRE) costs, FPGAs serves as an ideal solution for low-cost, low-to-mid volume, but power conscious embedded designs [18]. This makes the FPGA power consumption a critical design constraint. Two key differences separate FPGAs from ASIC with respect to power consumption. One, the FPGAs have a lot of redundant structures to ensure mapping and routing flexibility of the designs which are simply not present in ASICs. The second key difference is that FPGAs make use of more transistors in comparison to implement the same logic in ASICs. Hence, the FPGAs consume more power for a similar operation. The main challenge here is to find some novel solutions to reduce power consumption on physical as well as on system level, e.g. finding the suitable granularity mix and datapath structures of the underlying heterogeneous blocks [16].

2.3. PROCESSING ARCHITECTURES

The three important processor architectures, CPU, GPU, and FPGA, which are essential for varying computational needs, are discussed in this section. The main characteristics, which are leading to the increased usage in many scientific fields such as image processing, security, etc., are elaborated. These architectures are central to the development of this thesis. The selected applications are executed on these architectures (CPU and GPU) to develop the conceptual model and later implemented in hardware (FPGA) to verify the model.

2.3.1. CPU

CPU is one of the important processors used in high-performance computing as well as in the daily use to run a variety of applications. The notable contributions to the evolution of modern CPUs are mainly due to Intel (tick-tock model [19]) and AMD. CPUs provide a fast response times to a single task and are designed for general-purpose usage in order to adapt well to a wide range of applications. Without a doubt, CPUs provide the best single thread performance. The crucial features, which are integrated into the modern processors, are summarized in Fig. 2.4. However, there is some price associated with these advances, mainly in the form of power consumption and complexity per given area. Due to these reasons, modern CPUs can only pack a handful of processors on the same die.

Modern applications contain different characteristics such as data level parallelism, thread level parallelism, task level parallelism, kernels with a lot of diverging branches, etc., which means that one fixed hardware architecture is not optimal for these applications. Features that are shown in Fig. 2.4 try to overcome various barriers involved in the application under consideration. Hierarchy of caches helps to exploit temporal and spatial locality of instruction and data accesses, multiple SIMD units helps to increase data-level parallelism, branch prediction improves the performance for control-intensive kernels, long pipelines increase throughput as well as frequency, multi-cores along with hyper-threading may increase task level parallelism as well as thread level parallelism to some extent, and other features such as Out-Of-Order (OoO) execution, speculative execution, super-scalar etc., are responsible for the additional throughput in performance of the application under consideration.

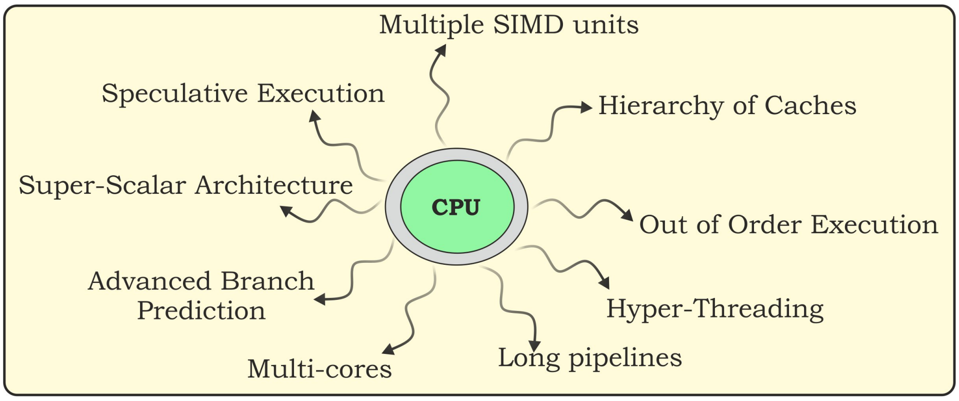


Figure 2.4: Advanced features of the modern CPU

2.3.2. GPU

Latest advances in the field of computing have led to a generation of massive amounts of data. Soon, the amount of digital data exceed exabytes (10^{18}) [20]. Working with this ever-growing data in a timely manner has made throughput computing an important factor for emerging applications such as database, games, video, finance, etc. These applications possess an ample amount of data level parallelism, and hence, the data can be independently processed and in any order on different processing elements for a similar work-related kernels such as filtering an image, aggregating, ranking, etc. This feature combined with a processing deadline defines *throughput computing applications* [21]. GPU is, undoubtedly, one of the best platforms for throughput computing applications.

GPU was originally designed for graphics processing with small processing elements. Later on, with the addition of sophisticated programming framework, programmers started exploring the computation of general purpose applications, which led to the rise of General Purpose GPU (GP-GPU) computing. NVIDIA was the first company to introduce the programming framework, called CUDA (Compute Unified Device Architecture), and ever since they are the market leader in the development of high-performance GPUs. GPUs are built specifically to handle a large degree of data parallelism which is easily available in many important applications such as image processing, graphics rendering, simulation modeling, etc.

Fig. 2.5 summarizes the features present in most of the latest GPUs. Unlike CPUs, GPUs have a limited memory size with 2-level memory hierarchy and a high single thread latency. Typical NVIDIA GPUs have about a few Giga-Bytes (GB) of global memory and contain several Streaming Multiprocessors (SMs). Each SM has a few Kilo-bytes of shared memory (around 32KB) and L1 cache (around 16KB), 32 cores, 2 Warp Scheduler and dispatch unit, a bunch of registers acting as a fast memory and Special Functional Units (SFUs). With respect to programming framework, millions of threads can be created and a GPU scheduler dispatches the threads to the available cores and has the ability to switch between the threads when long latency events such as memory accesses are encountered. Data transfers with CPU can be made asynchronous to keep the CPU busy with computations without waiting for transfer completion. Barriers provide support for synchronization, and atomic execution is used to avoid race conditions. Task level parallelism can be achieved by executing multiple kernels in parallel.

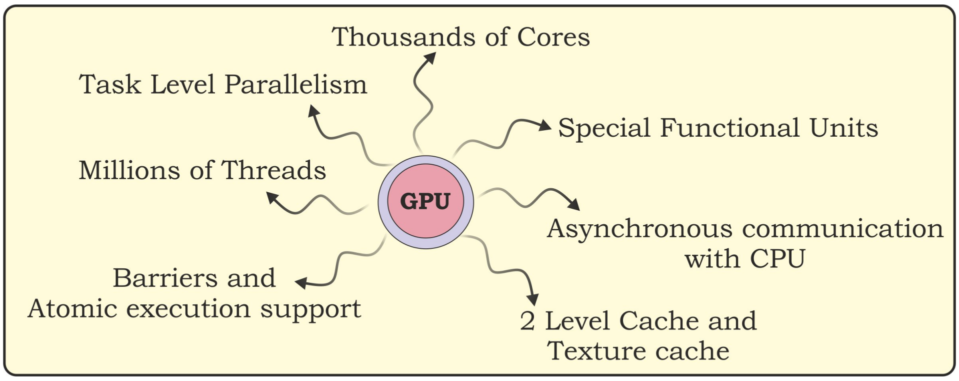

Thousands of Cores
Task Level Parallelism
Special Functional Units
Millions of Threads
GPU
Asynchronous communication with CPU
Barriers and Atomic execution support
2 Level Cache and Texture cache

Figure 2.5: Features of the modern GPUs

2.3.3. FPGA

In the last two subsections, we came across two most important fixed hard processor designs, CPU and GPU. However, there exists a multitude of applications which demands the change in underlying hardware as the hardware designs constantly evolve over time. This is where a programmable hardware comes in. Three types of programmable logic are defined and are shown in Fig. 2.6. Simple PLDs consist of either Programmable Array Logic (PAL), which is a combination of Programmable AND array and fixed OR array, or Programmable Logic Array (PLA), which is a collection of both Programmable AND array and Programmable OR array. CPLDs and FPGAs provide the highest density, sophisticated and most advanced programmable logic devices. CPLDs offer lower cost, provide more I/O in a small space, have faster and more predictable timing properties while FPGAs offer the highest gate densities and more features [22]. FPGA is focused on in this thesis as the hardware is synthesized for it.

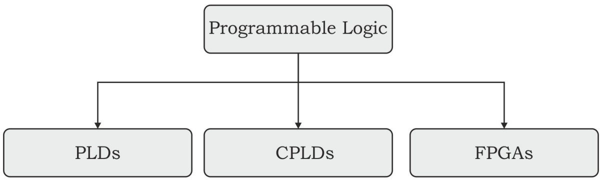


Figure 2.6: Types of Programmable logic. (PLD : Programmable Logic Devices, CPLD : Complex PLD, FPGA : Field Programmable Gate Array)

FPGA is one of the key technology enabling the development of many novel products that includes secure data centers and cloud computing, robotics, machine vision and learning, autonomous vehicles, the Internet of Things, renewable energy, home automation, 8K video and video surveillance, facial recognition and bioinformatics, 5G cellular networks, smart medical diagnostics, etc., to name a few. FPGAs are displacing ASICs in a rapid pace as the time to market is faster and the capability provided is nearly the same. The latest ones contain as many as 20 million gates. They are becoming increasingly prevalent in most electronic items. Programmable logic devices now constitute a $6 billion a year business, and are expected to grow to a whopping $10 billion a year by 2020 as part of hundreds of billions of dollars in finished products [23]. They provide great flexibility in hardware designs allowing to synthesize almost any digital function, and combined with hardcore Intellectual Properties (IPs) such as Digital Signal Processor (DSP), Processor cores, etc., these devices make way for a quick development at relatively low cost.

FPGA BASIC ARCHITECTURE

FPGA is made up of three basic elements as shown in Fig. 2.7. Programmable logic blocks, also called Configurable Logic Blocks (CLBs) implement the necessary logic functions; programmable input/ouput (I/O) blocks help in making off-chip connections; and programmable interconnect help in routing various signals among CLBs as well as I/O blocks.

Programming Technology: Number of programming technologies are used for FPGA recon-

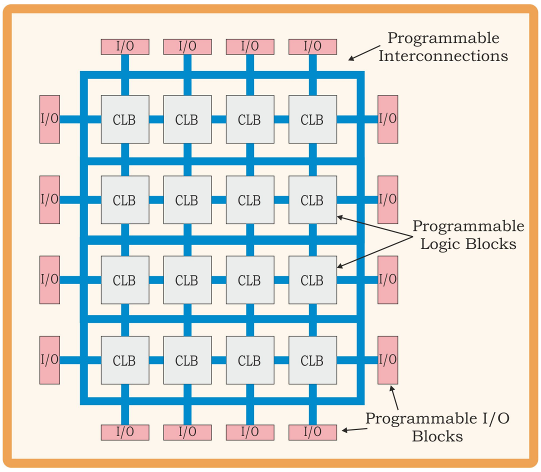


Figure 2.7: Basic internal architecture of FPGA

Table 2.2: Three prevalent FPGA programming technologies along with their trade-offs.

Property/ Device Type	SRAM	Flash	Anti-Fuse
CMOS Process	Standard	Non-Standard	Non-Standard
Programmability	Highly Programmable	Limited Programmability	Non-Programmable
Area Cost	High Cost	Medium Cost	Low Cost
Memory Retention Type	Volatile	Non-Volatile	Non-Volatile

figuration. The prominent ones are listed along with their characteristics in Table 2.2. An SRAM cell requires 6 transistors, making the SRAM based programming technology costly in terms of area compared to other two technologies. However, as SRAM uses the standard CMOS process, it leads to increased integration, lower dynamic power consumption and higher speed. Ideal programming technology would provide a re-programmable, non-volatile, efficient area cost and uses the standard CMOS process. However, none of the technologies mentioned in the Table 2.2 satisfy these conditions. SRAM based programming technology is widely used as it uses standard CMOS process, and it is expected to continue to dominate the other two programming technologies [24].

CLB: It is a basic building block of FPGA which provides the logic and storage functionality for a design under consideration. In the extreme ends, this component can be either a transistor or an entire processor . Very fine-grained CLBs like a transistor require a large amount of programmable interconnect which eventually results in an FPGA that suffers from area-inefficiency, low performance and high power consumption, whereas on the other hand (in case of processor), the basic logic block is very coarse-grained and cannot be used to implement small functions as it will lead to wastage of resources [24]. However, there exists a spectrum of basic logic blocks that exist in between these two extremes and are made up of NAND gates, an interconnection of multiplexors [25], lookup table (LUT) [26] and PAL style wide input gates [27]. LUT-based CLBs are used by the commercial vendors such as Xilinx and Altera to provide basic logic and storage functionality. They provide a nice trade-off between the two extremes (too fine-grained and too course-grained). CLBs are implemented using SRAMs, Multiplexers and D-type flip-flops.

Routing Architecture: Programmable routing network connects various logic blocks and I/O blocks to implement any user-defined circuit. The routing interconnect is made up of wires and programmable switches, which are in turn configured using the programmable technology. The routing network of an FPGA occupies 80—90% of total area, whereas the logic area occupies only 10-–20% area [28]. Since FPGA portray to be an important architecture to implement any user-defined function, its routing interconnect must provide deep flexibility in accommodating a wide

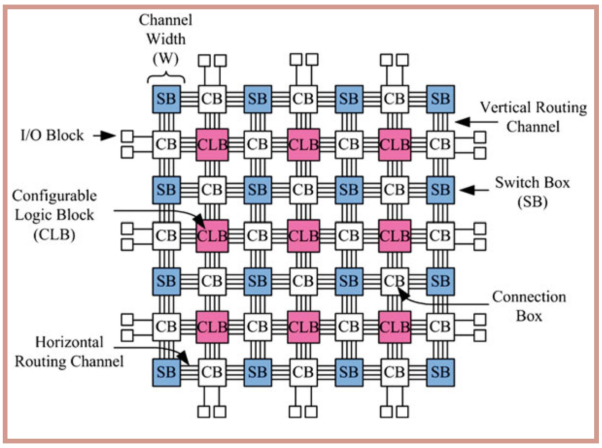


Figure 2.8: Island Style or mesh-based FPGA routing architecture [28].

variety of circuits with different routing demands. Routing network needs to provide a balance in flexibility and efficiency in connecting short path circuits as well as some distant connections so that the overall efficiency of FPGA is not compromised. Two most important routing architectures considered among commercial vendors are hierarchical [29] or island-style [28] architecture.

Fig. 2.8 shows the Island-style or mesh-based routing architecture. It is called so as the logic blocks look like islands in a sea of routing interconnect. Switch Boxes (SB) connect horizontal and vertical routing tracks, and Connection Boxes (CB) connect various logic blocks (CLBs). Altera's Stratix II [30] is one of the architectures which uses this type of routing.

Hierarchical routing architectures exploit the fact that most logic designs exhibit locality of connections by dividing FPGA logic blocks into separate groups/clusters. A hierarchical connection is formed by recursively connecting the clusters. Connections between logic blocks within same cluster are made by wire segments at the lowest level of hierarchy whereas the connection between blocks residing in different groups requires the traversal of one or more levels of hierarchy [24]. The signal bandwidth is widest at the top-level hierarchy, and it varies as we move to bottom-level hierarchy. Altera Flex 10K [31] is one of the architectures which has this type of routing.

EVOLUTION OF VIRTEX FPGAS

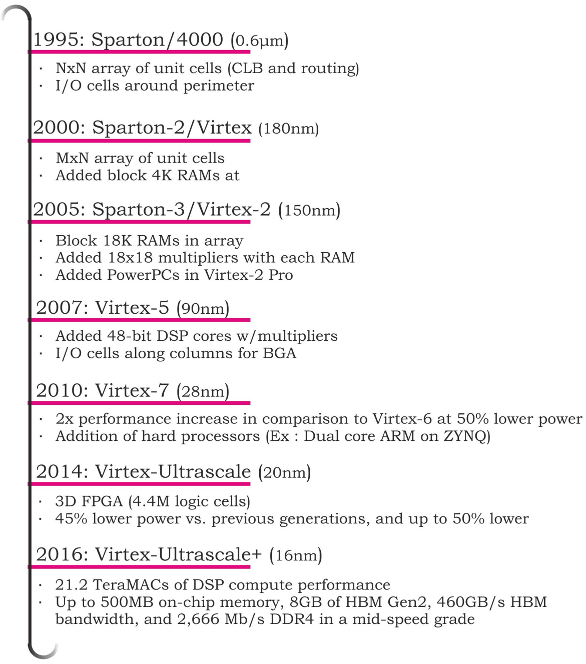
Altera and Xilinx are the industry leaders in the innovation and development of FPGA. In this thesis, Xilinx FPGA, and in particular Virtex-7 type FPGA, is utilized. In this regard, it makes sense to discuss a brief history on the development of Virtex series FPGA. Virtex belongs to the flagship family of Xilinx FPGA products. Other ones include kintex (mid-range), Artix (low-cost) and Sparton (low-cost) series.

Fig. 2.9 shows the highlights in the last two decades of the Virtex series evolution. In 1995, Xilinx developed an array of NxN cells with routing along the center lines. Block 4k RAMs were added by 2000. Addition of hard blocks such as multipliers and PowerPC processor were done in 2005. In 2007, DSP cores were added. Virtex-7 type FPGAs were introduced in 2010. The notable Reduced Instruction Set Architecture (RISC) based processor called Advanced RISC Machine (ARM) was added to create a System On Chip (SOC), which is the FPGA used in this thesis. 3D-FPGAs, called Virtex-Ultrascale, were created in 2014 which have the capability to accommodate in excess of 4 Million logic cells. Virtex-Ultrascale+, the highest performance FPGA in this series was introduced in 2016. It is clear from this evolution that FPGAs are not just a glue logic but are becoming subsystems or a complete system solution for various academic or commercial problems.

SELECTION CRITERIA FOR FPGAS

A number of FPGA vendors produce a different type of FPGAs suitable for certain type of applications. It is therefore essential to know various capabilities of the FPGA under consideration. Below are the main criteria for selecting FPGAs [23].

- Reprogrammablity (Configuration Memory Type): As discussed in the previous subsections, SRAM and FLASH are reprogrammable, and Anti-fuse is one time programmable.
- Size or Logic Density: Amount of logic in systems gates, Logical Elements (LEs), slices, macrocells, etc.
- Cost per logic gate: Cost per logic gate plays an important role as the cost of the device might exceed the design budget with an increase in the logic density.
- Speed: Processing speed, which is measured by the maximum clock frequency, is an essential feature that can determine the limit of the highest performance one can get on a given FPGA.
- Power Consumption: This contributes to both static (Idle Power) and dynamic (Application runtime) power consumption which is directly proportional to the processing frequency.



1995: Sparton/4000 (0.6µm)
- NxN array of unit cells (CLB and routing)
- I/O cells around perimeter

2000: Sparton-2/Virtex (180nm)
- MxN array of unit cells
- Added block 4K RAMs at

2005: Sparton-3/Virtex-2 (150nm)
- Block 18K RAMs in array
- Added 18x18 multipliers with each RAM
- Added PowerPCs in Virtex-2 Pro

2007: Virtex-5 (90nm)
- Added 48-bit DSP cores w/multipliers
- I/O cells along columns for BGA

2010: Virtex-7 (28nm)
- 2x performance increase in comparison to Virtex-6 at 50% lower power
- Addition of hard processors (Ex : Dual core ARM on ZYNQ)

2014: Virtex-Ultrascale (20nm)
- 3D FPGA (4.4M logic cells)
- 45% lower power vs. previous generations, and up to 50% lower

2016: Virtex-Ultrascale+ (16nm)
- 21.2 TeraMACs of DSP compute performance
- Up to 500MB on-chip memory, 8GB of HBM Gen2, 460GB/s HBM bandwidth, and 2,666 Mb/s DDR4 in a mid-speed grade

Figure 2.9: Brief note on the evolution of Xilinx Virtex Series FPGA

- I/O Density: Total number and type of I/Os present add significant value to FPGA. In addition, Cost per I/O also needs to be considered.

- Hard IP available on-chip : This is one way of increasing FPGA values by integrating hard IPs such as memory, DSP Blocks, transceivers, processors etc.

- Deterministic timing: Timing needs to be consistent in every implementation of the design.

- Reliability: This is an important concern in safety-critical designs and is measured by Failures In Time (FIT) and Mean Time Between Failures (MTBF).

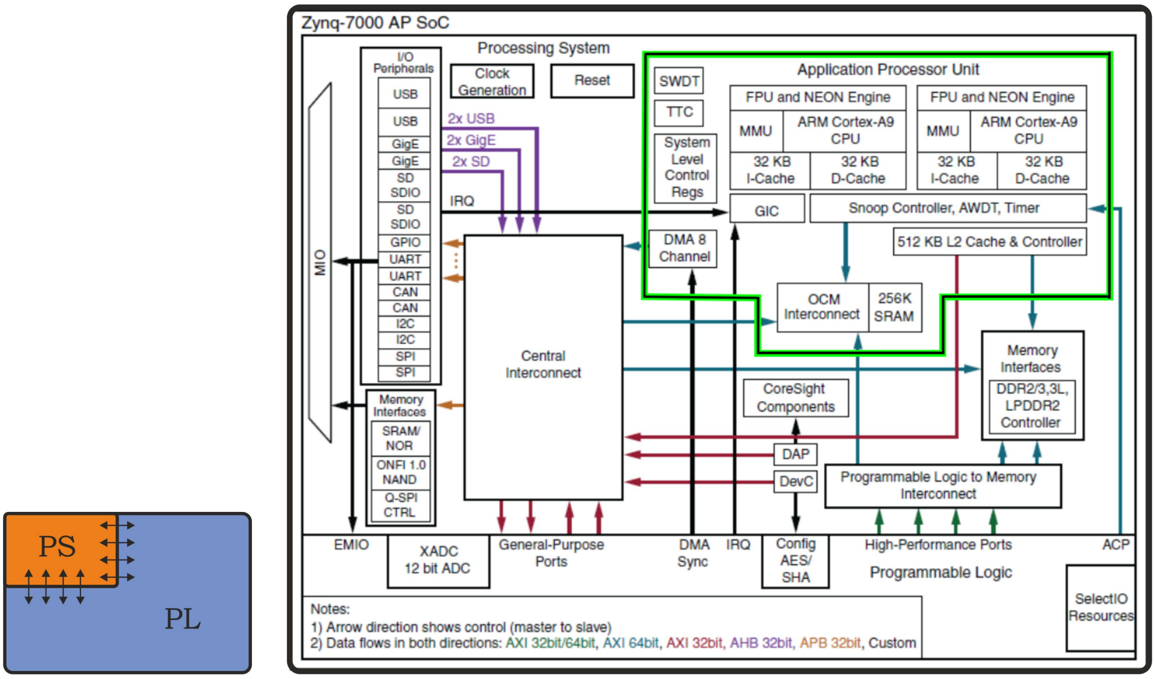
- Endurance: It is an important parameter to be concerned for FLASH based FPGAs as well as the design for long lifetime products.

- Data Security: It is quite an important ability that is becoming increasingly relevant to protect the design information in the FPGA bitstream as well as the data traveling through the FPGA.

ZYNQ FPGA

Zynq FPGA is an SoC which combines a dual-core ARM Cortex-A9 application grade processor capable of running full operating systems like Linux, with traditional FPGA logic fabric based on Xilinx 7-series FPGA architecture. The SoC solution is of lower cost, enables faster and more secure data transfers between the various system elements, has higher overall system speed, lower power consumption, smaller physical size, and better reliability [32]. This FPGA is extensively used in this thesis to create a dynamic platform, capable of switching two or more softcores of different capabilities, and profile suitable applications on it. In particular, PYNQ-Z1 board, which houses the Zynq FPGA along with some additional peripherals such as DDR Memory (512MB), is used.

Fig. 2.10(a) shows the high-level model of the Zynq FPGA. Processing System (PS) and Programmable Logic (PL) are the two main parts of the Zynq FPGA. These two parts are connected with the standard Advanced eXtensible Interface (AXI) based interfaces, which provide high bandwidth



(a) (b)

Figure 2.10: (a) Top level view of Zynq FPGA (b) Block level view of Zynq Processing System [32]

and low latency connections. Fig. 2.10(b) depicts a detailed block diagram of the Zynq PS architecture. It consists of the Application Processing Unit (APU) and peripheral interfaces such as cache memory, memory interfaces, interconnect, and clock generation circuitry. The APU is primarily comprised of two ARM processing cores, each with associated computational units: a NEON™ Media Processing Engine (MPE) and Floating Point Unit (FPU); a Memory Management Unit (MMU); and a Level 1 cache memory (in two sections for instructions and data) [32]. Furthermore, the APU also contains L2 cache memory and On-Chip Memory (OCM). A Snoop Control Unit (SCU) constitutes a bridge between the L2 cache, OCM memories, and the ARM cores. The PL receives four clock signals, each can range up to 250MHz, from PS.

2.4. SELECTION OF SOFTCORES

Researching the improvements of future manycore systems is mainly made using system simulation approaches, where a lot of tools are available. Nevertheless, when it comes to the topic of prototyping, hardware designers depend on a platform on which they can realize their designs. To build such a platform from scratch is a complicated task and very time-consuming. In this regard, for implementing a conceptual model in hardware, existing relevant softcores are first researched. There are quite a lot of open-source softcores available with a different set of capabilities. Selection of suitable softcores is crucial to the development of this thesis. In this regard, a brief note on some of the relevant softcores required for sequential and parallel needs of the applications are presented in this section.

2.4.1. SINGLE-CORE SOFTCORES

In this subsection, single-core (or dual-core) softcores are discussed. First, a brief overview of the important softcores is given. Then, the analysis is made to select the softcore for hardware implementation.

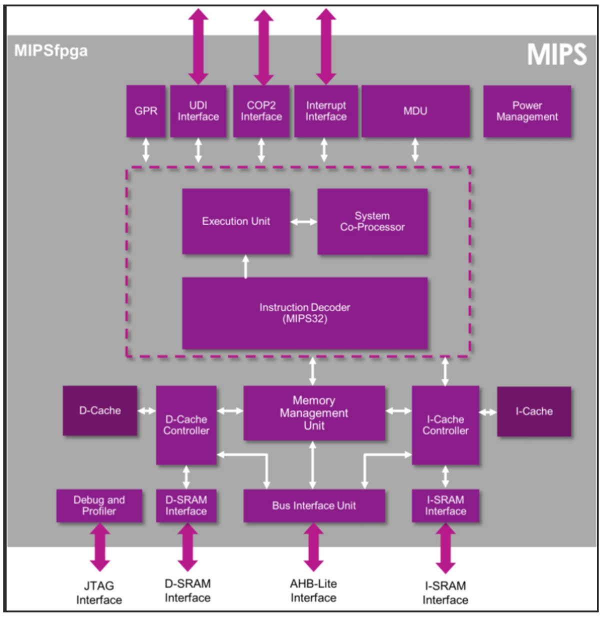


Figure 2.11: A block diagram view of MIPSfpga core [33]

MIPS

MIPS (Microprocessor without Interlocked Pipeline Stages) is a RISC based instruction set architecture (ISA), which was developed by MIPS Technologies three decades ago. Initially started as a research project at Stanford University, it has now become a revolutionary product used in many areas such as Artificial Intelligence (AI), Internet of Things (IOT), Automotive domain, Networking, etc.

The highest performance core in the MIPS cores, called R10000, has a superscalar architecture and has a capability of OoO execution. But this core is not available as an open-source softcore. Like this, most of the MIPS processors are commercially based. However, they have released one version called MIPSfpga as an open-source project a few years ago for academic purposes. The architecture of MIPSfpga is shown in Fig. 2.11. MIPSfpga is essentially the microAptiv core [33] targeted to FPGA. It has a single issue 5-stage pipeline, MMU, and a 2-way set associative cache. This is the simplest of the MIPS processor cores.

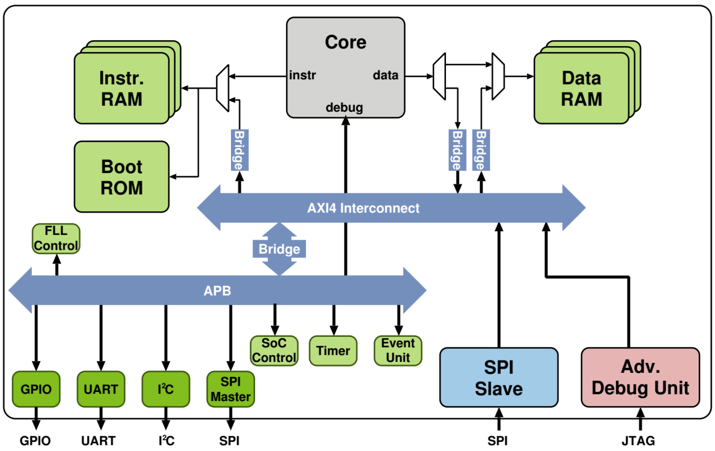


Figure 2.12: Overview of Pulpino archtecture [34]

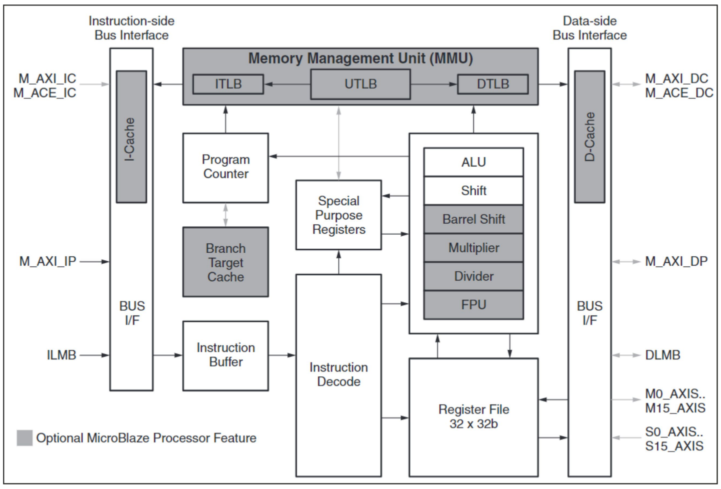


Figure 2.13: A block diagram view of Microblaze softcore [35]

PULPINO

Pulpino is an open source single-core microcontroller system, based on a 32-bit RISC-V core and can be configured to use with either the RISCY or the zero-riscy core [34]. Fig. 2.12 shows the block diagram of the pulpino system. It uses AXI interconnect along with a bridge to connect Advanced Peripheral Bus (APB) for simple peripherals shown in the referred figure. It does not include cache but it has two RAMs (Instruction and Data RAMs) that can be accessed via advanced debug unit.

A core is either the RISCY or the zero-riscy core. RISCY is an in-order core consisting of a 4-stage pipeline and has an Instructions Per Cycle (IPC) close to 1. Zero-riscy is a smaller version of the core which is also an in-order core with the 2-stage pipeline. This core is designed for applications with ultra-low power and ultra-low area constraints. Both cores support compressed instructions (RV32C) and low power mode, wherein the Pulpino system can be put in to low power mode when the core is idle and can be woken up by any event/interrupt.

MICROBLAZE SOFTCORE

Microblaze is an industry-standard softcore processor from Xilinx. It is a RISC based DLX architecture (a modified version of MIPS CPU) described in [13]. It has a versatile interconnect system consisting of the primary I/O bus being the AXI interconnect to connect various peripherals and a Local Memory Bus (LMB), a dedicated bus which provides access to fast on-chip storage.

Fig 2.13 shows the block diagram of the Microblaze system. It is a highly configurable system. Many parameters are user configurable: cache size, pipeline depth (3-stage/5-stage), MMU and various embedded peripherals. It gives multiple predefined configurations such as minimum area, maximum performance, maximum frequency, Linux with MMU, etc. It also includes branch prediction mechanism using the branch target cache synthesized as Block RAM (BRAM) in an FPGA.

ρ-VEX SOFTCORE

The ρ-VEX is a multi-issue reconfigurable architecture, taken originally from Hewlett-Packard (HP) and STMicroelectronics, and is developed extensively as a part of the overall “Liquid Architectures" research theme within the Computer Engineering Lab at TU Delft [36]. Unlike other softcores previously discussed, this softcore is based on a Very Long Instruction Word (VLIW) architecture. The ISA of ρ-VEX (VLIW Example) is derived from the VEX ISA [37].

Fig. 2.14 illustrates the 32-bit, 4-issue ρ-VEX VLIW processor. The computation pipeline consists of fetch, decode, execute and writeback stages. Execute stage contains parallel Arithmetic logic units (A) and Multiplier (M) units, the branch control (CTRL) unit and load/store (MEM) units. CTRL

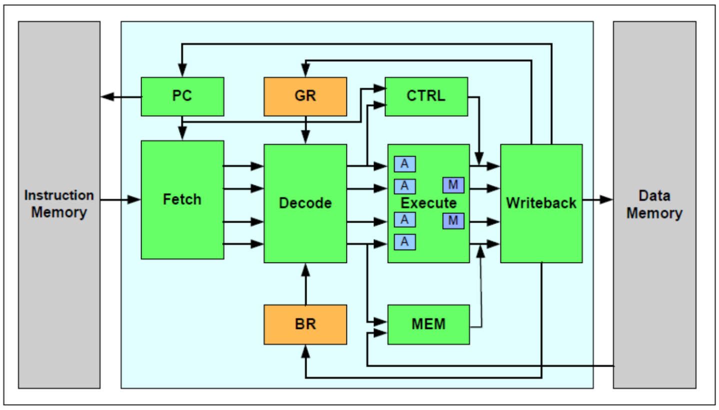


Figure 2.14: A block diagram view of ρ-VEX 4-issue VLIW Processor [38]

block handles the branching operations, and MEM unit controls the load/store operations. All execution unit operations are single-cycle based, except for the MEM and MUL units that take one extra cycle. It has a forwarding logic to minimize pipeline stalls. It supports L1 instruction and data caches that are implemented in BRAMs. Further, a shared L2 cache and the ability to run the applications from the DDR memory is provided in [39] running around 50Mhz frequency.

DISCUSSION OF THE SOFTCORE SELECTION:

The discussion for selecting a previously examined single-core softcores is done here. The main criteria for selecting the softcore can be derived from Fig. 2.4. The big core which can utilize an entire FPGA area for doing useful work for the application under consideration is the desired single-core softcore. Open-source softcores supporting as many advanced features of the modern CPU such as huge caches, branch prediction, OoO execution, long pipelines, high frequency, etc., are given more preference.

Table 2.3 summarizes the softcores discussed in this subsection. MIPS processor, provided as open-source, is a very small processor and does not provide any of the modern CPU features discussed earlier. Pulpino is a very sophisticated processor with 4-stage pipeline and FPU, but it is still quite small and doesn't support caches. Microblaze is a relatively large processor supporting FPU, branch prediction, caches along with 5-stage pipeline. It doesn't have the ability to run from DDR, so its memory is limited to 128KB on-chip memory, a maximum one supported from Xilinx. This can be considered for a single-core softcore. This leaves us with ρ-VEX softcore. The ρ-VEX, a VLIW based architecture, is an attractive alternative to the RISC based softcores. It is capable of having 2/4/8-issue execution units supporting two-level caches and last but not the least; it has an ability to run applications from DDR. Furthermore, L2 cache supports multiple cores. 4-issue softcore is considered here as the 8-issue softcore cannot be synthesized in the limited space available on Zynq FPGA. This means the dual-core ρ-VEX coupled with multi-issue (4-issue) and two level caches can be synthesized to cover most of the Zynq FPGA space. Hence, ρ-VEX is also considered as a potential single-core softcore. In summary, the *Microblaze* and the *ρ-VEX* softcores are considered to realize

Table 2.3: Summary of the Single-Core Softcores

Soft-Core	Size	Sophisticated	Caches	Frequency (Mhz)	Special Features	Chosen?
MIPS	Small	×	✓	~100	5-stage Pipeline	×
Pulpino	Small	✓	×	~50	FPU, IPC~1, 2-stage or 4-stage Pipeline	×
Microblaze	Large	✓	✓	100-400	FPU, Branch Prediction, 3-stage or 5-stage Pipeline	✓
ρ-VEX	Very Large	✓	✓	50	Multi-issue, 2-level cache, dual-core, DDR support	✓

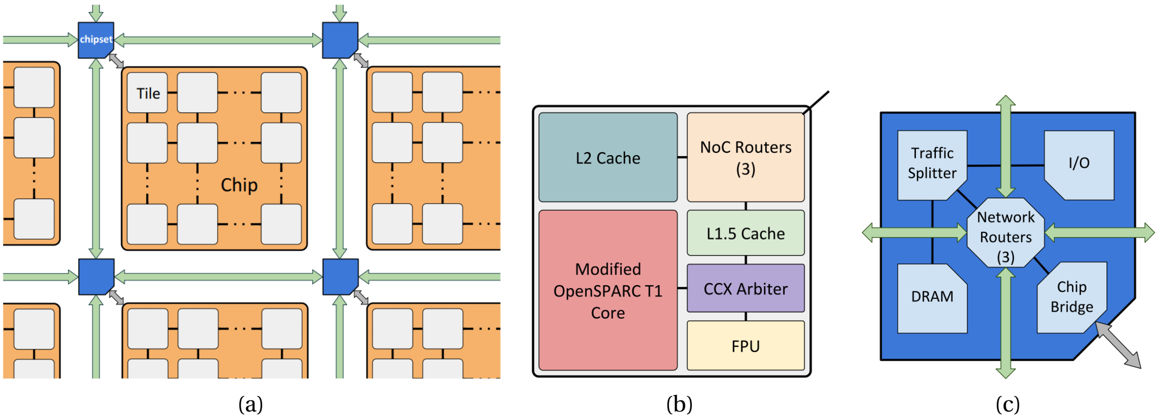


Figure 2.15: OpenPiton : (a) Overall architecture (b) Tile architecture and (c) Chipset architecture [40]

a single-core softcore.

2.4.2. MANY-CORE SOFTCORES

This subsection emphasizes the discussion related to manycore softcores. As followed in the last subsection, first brief information on the manycore softcores is given. Then, a trade-off among the considered softcores are discussed to choose the one for hardware implementation.

OPENPITON SOFTCORE

OpenPiton is a general-purpose multi-threaded, highly-configurable, scalable, tiled-manycore processor and framework [40]. It builds on the industry standard OpenSPARC T1 core but sports a completely scratch-built uncore. Cores are theoretically scalable up to 500 million. It has an exhaustive test infrastructure with over 8000 tests and can run full-stack multiuser Debian Linux.

Fig. 2.15(a) shows the tiled manycore design of a 64-bit OpenPiton architecture. Tiles are connected via three Networks On-Chip (NoC) routers in a 2D mesh topology as shown in Fig. 2.15(b). Three level cache hierarchy is provided, with L1 and L1.5 caches and a distributed, shared L2 cache following a directory based MESI [41] cache coherence protocol. Instead of modifying the original L1 cache, designed for OpenSparc T1 core, to alleviate the issues faced such as the requirement of high write-bandwidth due to the existence of write-through caches, an extra level called L1.5 was introduced. Each tile also has an optional dual-precision Floating Point Unit (FPU). A Chipset which is shown in Fig. 2.15(c) consists of I/O, DRAM controllers, chip bridge, traffic splitter, and inter-chip network routers. All the components in a chipset work to handle traffic with inter-chip network routers.

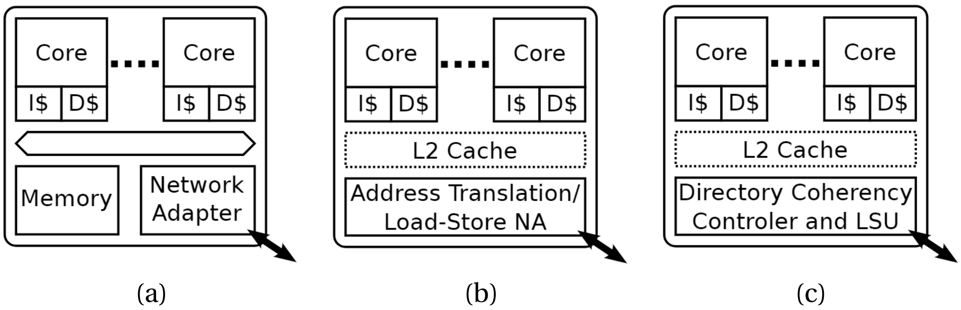

Core
Core
I$
D$
I$
D$
Memory
Network Adapter
L2 Cache
Address Translation/ Load-Store NA
Directory Coherency Controler and LSU
(a)
(b)
(c)

Figure 2.16: OpTiMSoC implementations : (a) Distributed memory (b) Partitioned global address space and (c) Shared memory [42]

OPTIMSOC SOFTCORE

Tile-based architecture, used in the previous softcore (OpenPiton), is a popular way to design many-core SOC as there exists a symmetry in the organization of processing elements which is scalable and suitable for hardware realization. Open Tiled Manycore System-on-Chip (OpTiMSoC) [42] also utilizes the tile-based design and has a distributed memory organization supporting the message passing programming interface. It has a LISNoC [43] as a central element, which is a free NoC implementation, mainly provided for academic or teaching purposes. The NoC is a packet-switched, wormhole forwarding, buffered implementation supporting virtual channels to avoid message dependent deadlocks [42].

Fig. 2.16 shows the different alternatives of OpTiMSoC implementation. Fig. 2.16(a) shows the distributed memory system in which each core is connected to locally shared memory, but due to restricted memory sizes, the prototyped version contains the partitioned global address space variant shown in Fig. 2.16(b), where each core accesses the chunk of global memory assigned to one tile. The shared memory based manycore implementation depicted in Fig. 2.16(c) is part of the future OpTiMSoC research. The manycore system currently runs at the 50Mhz frequency. The programming framework currently supports some of the message-passing Application Programming Interfaces (APIs) from the Multicore Association [44] such as the Multicore Communications API (MCAPI) and Multicore Task Management Working Group API (MTAPI).

OPENSCALE

OpenScale [45] is, once again, a variant of a tile based system which employs a distributed memory/message passing approach with Network Processing Unit (NPU) as its main component. Fig. 2.17 depicts the NPU [46] architecture, which includes a lot of pre-built components: (a) a SecretBlaze CPU, (b) an embedded RAM, (c) an interrupt controller, (d) peripherals such as timer and UART, (e) a Wishbone bus and (f) a HERMES-based router.

SecretBlaze CPU is a configurable open-source RISC softcore processor that implements the MicroBlaze ISA with a five-stage pipeline. It uses embedded RAM as a local memory and has a private caches that have no global cache coherency provided by OpenScale platform. It employs Message Passing Interface (MPI)-like API for message passing communication between the NPUs.

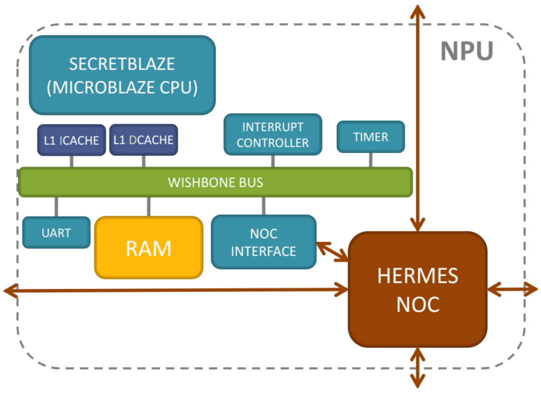


Figure 2.17: Architectural overview of Openscale NPU [45]

ρ-VEX STREAMING PLATFORM SOFTCORE

We have seen the ρ-VEX platform in the previous subsection. Here, we are considering a specially modified manycore version of ρ-VEX for streaming applications like image processing. Unlike other softcores discussed in this subsection, the ρ-VEX is based on VLIW architecture. The cores are cut down to a minimum size (2-issue cores) in order to efficiently utilize a given FPGA area to run many streams simultaneously. Each core contains their own instruction and data memories and are connected in a stream as shown in Fig. 2.18. A simple address decoder connects memory units to the

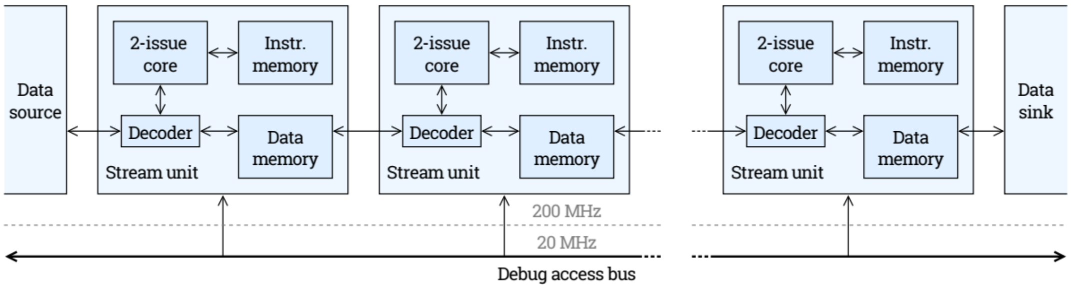


Figure 2.18: A single stream of RVEX two-issue cores [38]. Each core's memories and control registers are accessed through a low clock debug access bus to stop it from becoming timing-critical net.

processing core. Each core in a stream can access the memory of its predecessor and can write to a memory of a successor. The first and last cores in a stream are connected to DMA (Direct Memory Access) units to exchange data to external peripherals such as DDR.

DISCUSSION OF THE SOFTCORE SELECTION:

Here, a discussion is made to chose a suitable softcore required to implement manycore part of the dynamic platform. Fig. 2.5 shows the essential features expected out of manycore softcore. Cores should be small enough to fit as many as possible on the FPGA. Shared memory implementation is preferred over the distributed one due to the programming convenience. Special features such as barriers, atomic execution, SFUs, etc complete the manycore architecture.

Table 2.4 shows the highlights of the previously discussed softcores. OpenPiton is a sophisticated manycore softcore possessing a shared memory implementation for thousands of cores. L2 cache is a directory-based memory system capable of handling coherency for a large number of cores. However, the core is pretty big in size which means only a handful of them could fit on an FPGA which is not desired. Optimsoc is a distributed version of manycore softcore capable of running at the 50Mhz frequency. However, once again the core is large, and modification to the platform is a little complex process in the current version available. When the author was contacted regarding the complexity in the modification, he replied that in the future versions it would be made easy via XML file containing all the configuration details. OpenScale is also a tile-based manycore softcore with the distributed memory system. The tile-based softcores inherently increases the size of the softcore. OpenScale is no exception. Each core is embedded in an NPU consisting of a lot of components such as routers and NOC interfaces, which makes it large. Finally, ρ-VEX comes to the rescue. The cores are very minimal, the modification flexibility is high, the frequency is much higher than the other ones, and it was developed at TU Delft Computer Engineering Lab. Even though ρ-VEX softcore is not based on the shared memory architecture, it is the softcore considered to implement manycore softcore part of the dynamic platform as the current alternatives are not suitable.

Table 2.4: Summary of the Manycore Softcores.

Soft-Cores	Shared Memory	Size	Modification Capability/Flexibility	Frequency	Chosen?
OpenPiton	✓	Very Large	✓	67Mhz	×
XUM (Optimsoc)	×	Large	×	50Mhz	×
OpenScale	×	Large	✓	-	×
ρ-VEX	×	Small	✓	>100Mhz	✓

3

Conceptual Model

Big ideas more often than not sound stupid in the beginning.
- Brian Chesky (Airbnb co-founder)

In this chapter, conceptual modeling is done in order to identify the proposed architecture characteristics and trade-offs. First, the motivation for the concept development is explored in Section 3.1. Then, the platforms are introduced, and initial assumptions are explained in Section 3.2. In the same section, the kernels that are used in the modeling are explained. Further, the simulation model with real values is constructed to see how the proposed architecture fit into the model in Section 3.3. Subsequently, the abstract model is discussed in Section 3.4 to simplify the understanding, and overcome the limitation observed in Section 3.3.

3.1. Motivation

This section presents a motivation for the development of the conceptual model. It extends on the context discussed in Section 1.1. The authors in [1] extended the Amdahl's law to the multicore era. These laws are briefly revisited here for convenience.

The basic Amdahl's law, which states that if the fraction 'f' of the computation is infinitely parallelizable with no scheduling overhead, while the remaining fraction, (1-f), being totally sequential, then the enhanced speedup on 'n' processors is as given in Eq. 3.1. It is assumed that a multicore chip can accommodate at most 'n' chips, which are identical, and each core can provide a baseline performance. If 'r' cores are combined, then the performance of the resulting sequential core is perf(r), which is assumed to be $\sqrt{r}$. Three important laws, which are simple yet powerful, capturing three different multicore architectures, are discussed in [1].

Basic Amdahl's Law

$$\text{Speedup}_{\text{enhanced}}(f,n) = \frac{1}{(1-f)+\frac{f}{n}} \tag{3.1}$$

First law corresponds to the symmetric multicore chips as shown in Fig. 3.1 (a). Here, all the cores are of the same cost. Let us consider n=16 cores to fit on a multicore chip and combine every 4 cores to give 4 or (n/r) big cores of equal cost as shown in Fig. 3.1 (b). The speedup performance is given in Eq. 3.2. Comparing Eq. 3.2 to Eq. 3.1, the sequential part of the computation, corresponding to fraction (1-f), is improved by a factor perf(r), which is $\sqrt{r}$, and the parallel part uses all n/r cores in parallel to give a performance of perf(r)*(n/r).

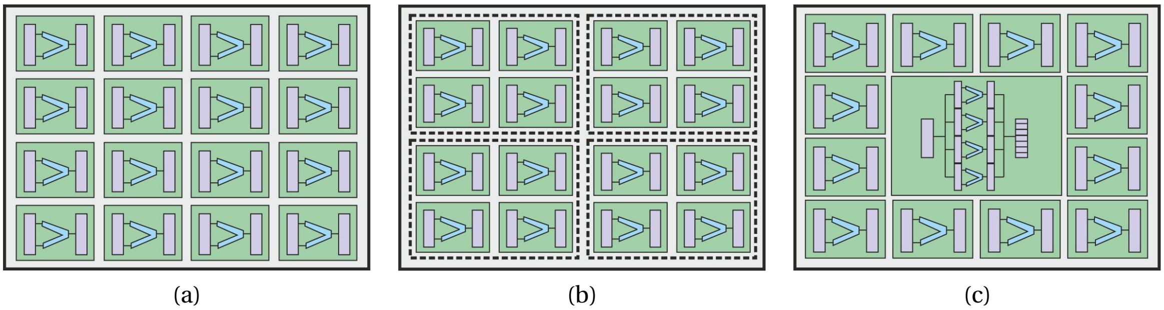


Figure 3.1: Amdahl's law in the Multicore era: (a) Symmetric Multicore with 16 base cores, (b) Symmetric Multicore with 4 big cores, each made up of 4 base cores, and (c) Asymmetric Multicore with one big core made up of 4 base cores and 12 base cores.

Amdahl's law to symmetric multicore chips

$$\text{Speedup}_{\text{symmetric}}(f,n,r) = \frac{1}{\frac{(1-f)}{perf(r)} + \frac{f*r}{perf(r)*n}} \tag{3.2}$$

The second law refers to the asymmetric multicore chips as shown in Fig. 3.1 (c). Let us again consider n=16 cores and combine 4 cores to give one big core as shown in the referred figure. The speedup performance is as mentioned in Eq. 3.3. Again, if this equation is compared to the basic Amdahl's law (Eq. 3.1), the sequential performance is improved by the factor perf(r) and parallel fraction, f, gets the performance of perf(r) from one big core and performance of 1 from the remaining (n-r) cores.

Amdahl's law to asymmetric multicore chips

$$\text{Speedup}_{\text{asymmetric}}(f,n,r) = \frac{1}{\frac{(1-f)}{perf(r)} + \frac{f}{perf(r)+n-r}} \tag{3.3}$$

The third law, which is the most relevant to this thesis, is shown in Fig. 3.2. In this scenario, the most optimistic case is considered wherein all the cores are combined to boost the performance of the sequential component, and all the base cores execute independently for the parallel portion of the computation. The resulting speedup is indicated in Eq. 3.4. This equation presents an ideal case in which the parallel portion is simultaneously executed by all 'n' cores and the sequential part is executed by one big core resulting from combining all the base cores dynamically without any overhead. This thesis, basically, builds on this law to identify the feasibility of implementing such architectures and understand its performance characteristics.

Amdahl's law to dynamic multicore chips

$$\text{Speedup}_{\text{dynamic}}(f,n,r) = \frac{1}{\frac{(1-f)}{perf(r)} + \frac{f}{n}} \tag{3.4}$$

3.2. Introduction to the Model

Several computational platforms were discussed in the previous chapter. For the development of the model, CPU and GPU platforms are selected. These platforms are chosen based on their unique characteristics.

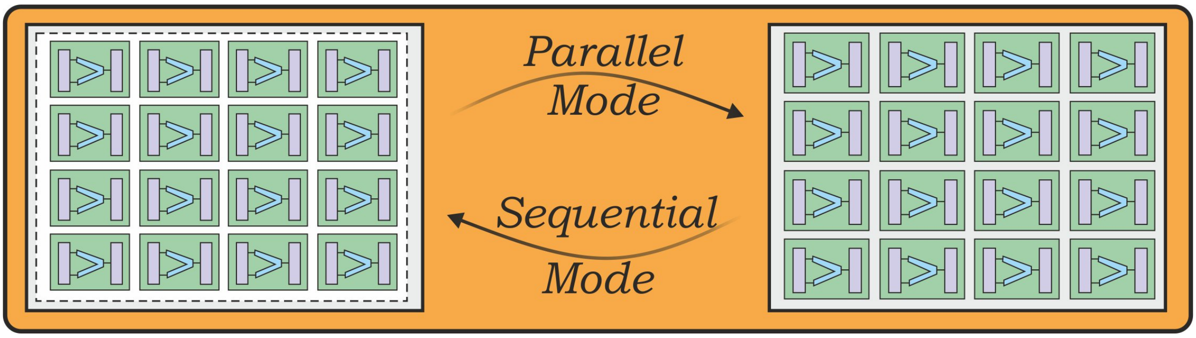


Figure 3.2: Dynamic Multicore Model of Amdahl's Law

CPU is one of the best available platforms to execute a multitude of applications without any hassles. It gets the unique characteristics from the innovations done on a single thread performance. Very high frequencies, Out of Order (OoO) execution support, huge caches, long pipelines, various SIMD units and the super-scalar architecture are the major features that differentiate it from other processors. One can easily conclude that it gives high performance for the sequential applications.

On the other hand, the GPU is one of the best platforms available to execute parallel applications. Some of its unique characteristics responsible for giving excellent multi-threaded performance are the availability of thousands of cores on a single die, two levels of caching, special functional units, medium to the high frequency range for cores and memory, and huge memory bandwidth to the number of active threads. This means that it has a potential to give the best performance for parallel applications.

For the purposes of constructing the model, it is assumed that the third platform, which is a *dynamic platform* (DP) built later in the implementation chapter, has a capability to switch between two different architectures (similar to CPU and GPU) without any overhead and in turn, providing the best of both platforms for the applications under consideration.

In the development of the conceptual model, we consider two kernels with extremely contrasting characteristics. One of the kernels is inherently sequential, which means the scope of parallelizing it is essentially low. The other kernel is extremely parallel and has the ability to take advantage of the parallelism offered by the platform under consideration. The two kernels are kept simple so as to not complicate the things in modeling and understanding the behavior. The expectation here is that if the kernel performs extremely well on one platform, say CPU; it should work very poorly on the other platform considered, say GPU, and vice versa. This expectation stems from the fact that most of the applications are neither extremely parallel nor sequential in nature, and if the extreme cases are chosen and averaged on both platforms with varying input sizes then it covers a spectrum of applications with varying levels of *granularity* in parallelism.

The sequential kernel does a vector addition with an intentionally induced dependency as given in Eq. 3.5. The vector is assumed to be consisting of only one component. The resulting vector ($\overline{V}_n$) is dependent on $\overline{A}_n$ and its previous instance, $\overline{V}_{n-1}$. Random number ensures that the dependency is preserved, and the work cannot be executed by more than one thread at a time.

$$\overline{V}_n = (\overline{V}_{n-1}) * \text{Constant} + \overline{A}_n * (\text{Random Number}) \tag{3.5}$$

The parallel kernel is constructed by considering a dot product of two vectors as shown in Eq. 3.7. The vector is composed of 8 sub-components as shown in Eq. 3.6, so that there is a reasonable computation involved. As there is no dependency between vectors, there exists huge parallelism to be utilized by any manycore platform.

Table 3.1: Details of the execution platforms used in the simulation.

	CPU	**GPU**
Name	Intel(R) Core(TM) i5-2410M CPU	Tesla K40c
Vendor	Intel(R) Corporation	NVIDIA Corporation
Compute Units	4	15
Clock Frequency	2300 MHz	745 MHz
Global Memory	5859 MB	11440 MB
Max Allocateable Memory	1465 MB	2860 MB
Local Memory	32768 KB	49152 KB
maxWorkGroupSize	8192	1024
maxWorkItemDim	3	3
maxWorkItemSizes(0)	8192	1024
maxWorkItemSizes(1)	8192	1024
maxWorkItemSizes(2)	8192	64
Device Version	OpenCL 2.1 (Build 10)	OpenCL 1.2 CUDA

$$\overline{V}_i = (\overline{A} + \overline{B} + \overline{C} + \overline{D} + \overline{E} + \overline{F} + \overline{G} + \overline{H}) \tag{3.6}$$

$$\overline{V}_3 = (\overline{V}_1.\overline{V}_2) \tag{3.7}$$

Firstly, performances for the chosen kernels are analyzed on the individual platforms. Then, the surface plot is created for varying combinations of granularity in parallelism for both sequential and parallel kernels. This means the vector size is varied for both applications on both platforms (CPU and GPU), and averages of all vector size combinations are considered as a source to the surface plot. For example, vector size of 1 Million for the sequential kernel is averaged with the vector size of 4 Million for the parallel kernel. This means the performances of all the platforms are projected on a plane with varying levels of parallelism. The dynamic model, as explained in this section, gives the best of CPU/GPU performances. In the final stage of the model, the performance of the DP is continuously decreased by a reduction factor to give a realistic scenario and observe the region where it performs better than CPU and GPU.

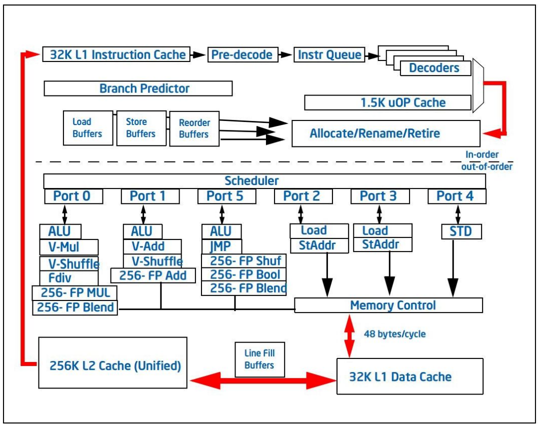
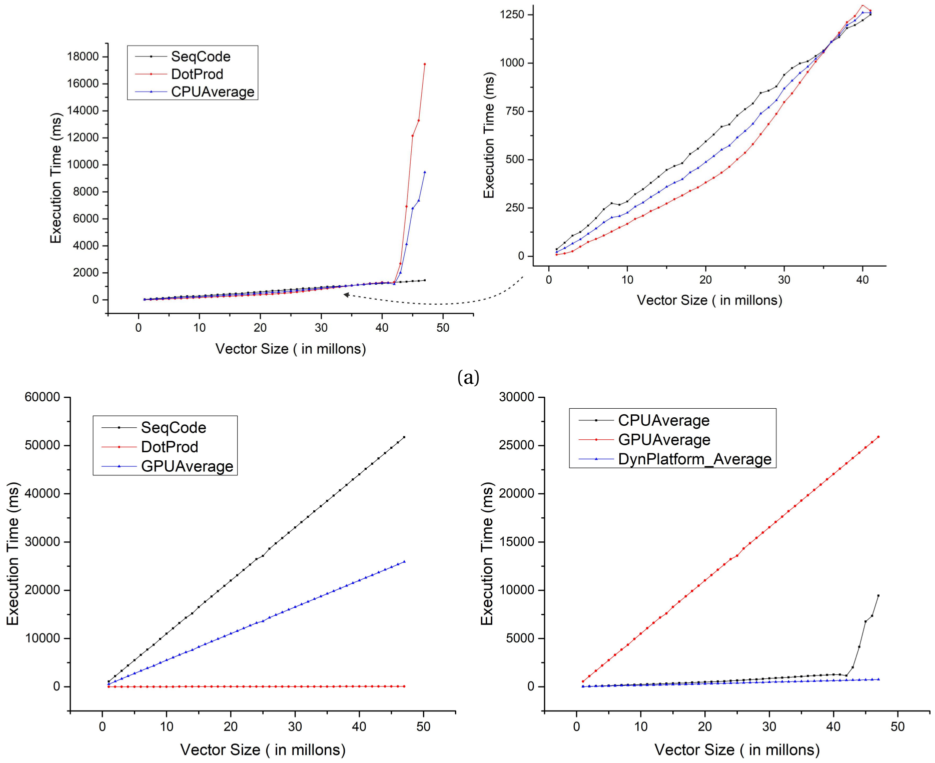


Figure 3.3: Intel's Sandy-Bridge CPU architecture for a single core

3.3. MODELING WITH REAL VALUES

In this section, the two kernels introduced in the previous section are simulated on both CPU and GPU. The DP takes the best of CPU or GPU performance values. Vector size is varied from 1 Million



(a)

(b)

(c)

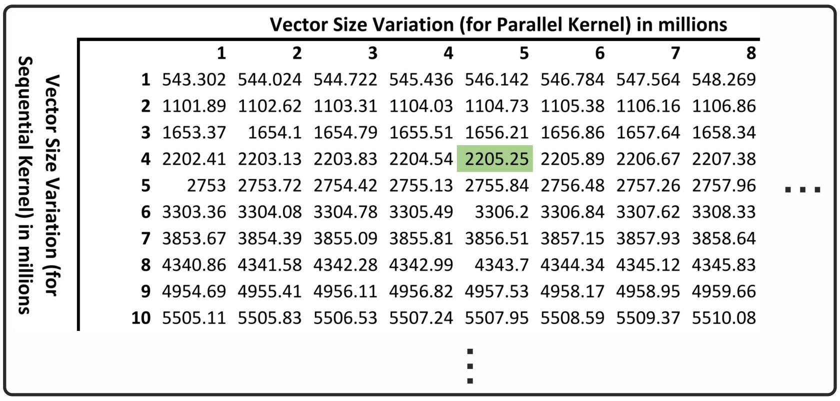
Figure 3.4: Performance of the sequential (SeqCode) and the parallel kernel (DotProd) kernels on (a)CPU, (b) GPU and (c) Dynamic Platform.

to 47 Million. The details of both the computational platforms used for modeling are given in Table 3.1. OpenCL is used as the programming language to execute on both CPU and GPU platforms.

Fig. 3.4 shows the variation of execution time for both the kernels on the CPU, GPU and the DP. For the sequential kernel, a single thread is created in OpenCL as it cannot be parallelized and for the parallel kernel, as many threads are created depending on the input vector size. Execution time, in case of GPU, corresponds only to the computation time as it is assumed that the transfer is done only twice (one from CPU to GPU and vice versa), and it does not contribute much to the overall time. Surface plots for all the three platforms are shown in Fig. 3.6. In these plots, the average performance for various values of vector sizes is calculated. To understand the surface plot easily, few values are shown in Fig. 3.5, corresponding to the GPU platform. The cell marked in green belongs to the average of performance of vector size of 5M for the parallel kernel and vector size of 4M for the sequential kernel. In this way, the average performances of all the combinations of the vector sizes for both kernels are projected onto a plane to get an insight into the behavior of the underlying platforms for varying levels of granularity in parallelism.

3.3.1. CPU

Fig. 3.4(a) shows the CPU behavior for both the considered kernels. For vector size till 41M, the sequential and parallel kernel seem to perform very closely and is therefore zoomed into that portion to see the difference more clearly. Though the expectation is that the sequential kernel should perform much better than the parallel kernel, the inverse behavior is observed at least in this considered range. The reason is two fold. The frequency of the CPU is 3 times that of the GPU (from Table 3.1), and it also contains many execution units. The CPU which is shown in Table 3.1 consists of 2 cores with 2-hyperthreads per core, and each core, in turn, contains 6 execution ports (as shown in Fig. 3.3), which provides a lot of parallelism that OpenCL can exploit. But once the RAM (Random



Vector Size Variation (for Parallel Kernel) in millions

Vector Size Variation (for Sequential Kernel) in millions

	1	2	3	4	5	6	7	8
1	543.302	544.024	544.722	545.436	546.142	546.784	547.564	548.269
2	1101.89	1102.62	1103.31	1104.03	1104.73	1105.38	1106.16	1106.86
3	1653.37	1654.1	1654.79	1655.51	1656.21	1656.86	1657.64	1658.34
4	2202.41	2203.13	2203.83	2204.54	2205.25	2205.89	2206.67	2207.38
5	2753	2753.72	2754.42	2755.13	2755.84	2756.48	2757.26	2757.96
6	3303.36	3304.08	3304.78	3305.49	3306.2	3306.84	3307.62	3308.33
7	3853.67	3854.39	3855.09	3855.81	3856.51	3857.15	3857.93	3858.64
8	4340.86	4341.58	4342.28	4342.99	4343.7	4344.34	4345.12	4345.83
9	4954.69	4955.41	4956.11	4956.82	4957.53	4958.17	4958.95	4959.66
10	5505.11	5505.83	5506.53	5507.24	5507.95	5508.59	5509.37	5510.08

...

⋮

Figure 3.5: Few values of the surface plot for understanding purpose. The values correspond to the surface plot in Fig. 3.6(b). The values (times in ms) are the averages of performance for the parallel kernel (towards right direction) and sequential kernel (towards downward direction) for varying vector sizes. The value marked in green correspond to the average performance of parallel kernel with vector size 5M and sequential kernel with vector size 4M.

Access Memory) is full, it cannot cope with the increasing vector size. This is why the execution time curve for parallel kernel starts to grow exponentially from 41M vector size, whereas the curve of the sequential kernel continues to grow linearly with vector size.

Fig. 3.6(a) shows the average performance of both kernels for various vector sizes on the 3D surface-plot for the CPU platform. The plane is divided into two parts. First part corresponds to the vector size from 0 to 41M and is almost constant, as explained in the previous paragraph. The second part starts from vector size 41M and grows almost exponentially due to the increased time spent on the parallel kernel.

3.3.2. GPU

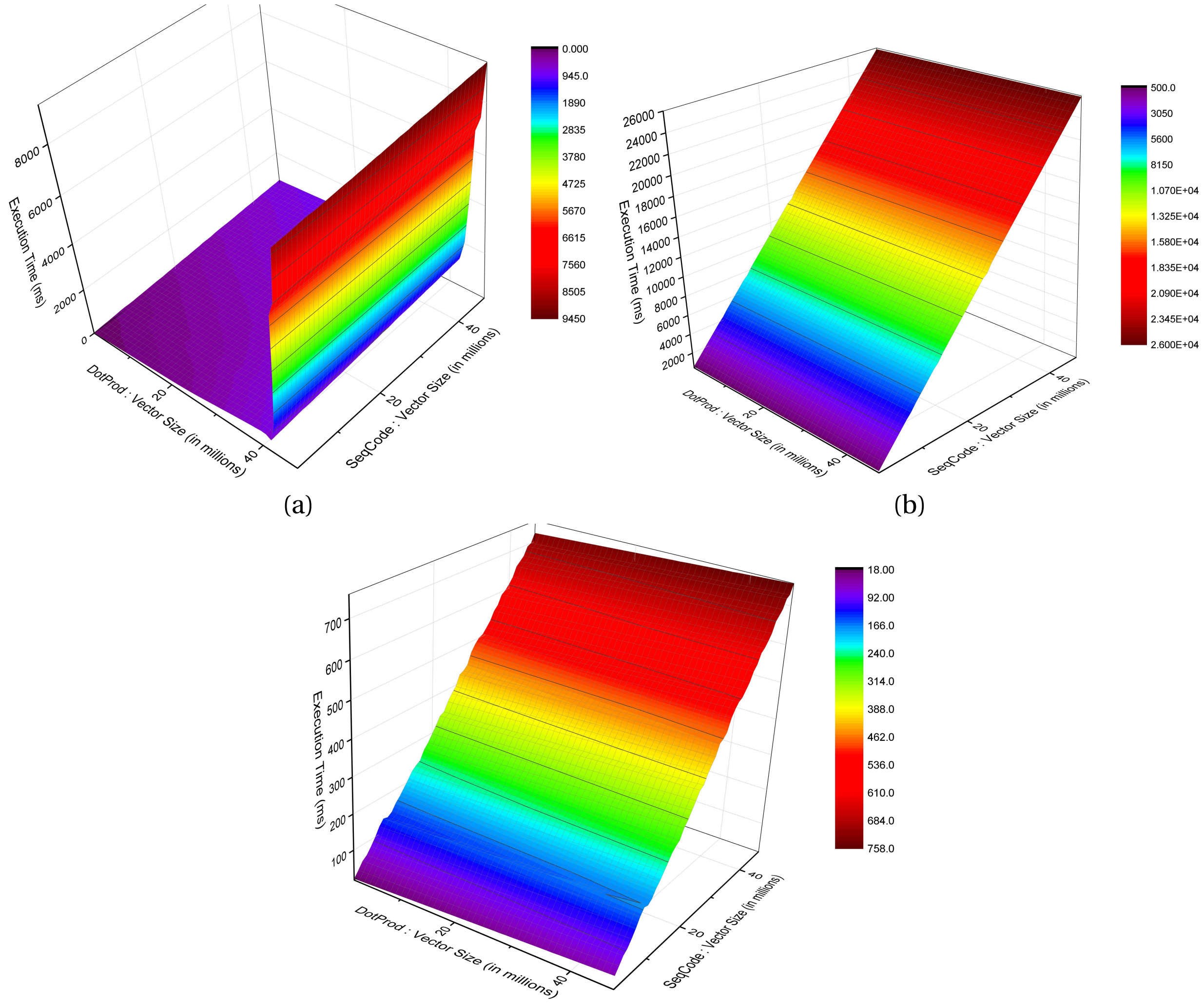
Fig. 3.4(b) shows the performance of the kernels on the GPU platform. GPU performs as expected with no surprises. For the parallel kernel, the execution curve remains within a small time range, while the time for sequential kernel increases quite linearly. It performs really well for the parallel kernel as it can handle huge amounts of parallelism and hides the individual thread latency by keeping all the cores busy. It behaves rather worse for the sequential kernel because only a single thread is active due to the dependency limitation and it cannot hide the long latency of the single thread as other threads are idle.

Fig. 3.6(b) shows the surface plot for GPU. The plane has a steep inclination in the direction of increasing values of vector size for the sequential kernel. However, the slope of the plane is almost zero towards parallel kernel. This trend indicates that the GPU is definitely not suitable for sequential applications and performs very badly due to the long latency of the single thread which otherwise would have been hidden by executing multiple threads simultaneously.

3.3.3. DYNAMIC PLATFORM

The performance on the DP, which takes the best of CPU/GPU values, is depicted in Fig. 3.4(c). For the obvious reasons, DP performs better than the CPU or GPU. But a closer look at the referred figure shows that the performance of the DP is closer to the CPU curve and highly distant from that of a GPU. The reason for this behavior is that the GPU performs rather badly for the sequential kernel than the CPU performs for the parallel kernel.

Fig. 3.6(c) shows the surface plot for the DP. The advantages of the DP can be clearly appreciated in this figure. Though the inclination towards the sequential kernel is steep when compared to the one in the parallel kernel direction, the values (execution times) are much smaller than the ones on GPU platform (Fig. 3.6(b)).



(a)

(b)

(c)

Figure 3.6: Surface plots depicting the average performances of the sequential kernel (SeqCode) and the parallel kernel (DotProd) for all combinations of vector sizes for (a)CPU, (b)GPU, and (c)Dynamic Platform

3.3.4. OBSERVATIONS

One of the key observations to be made here is that the average values of CPU for both kernels are never greater than those of GPU. This can be easily observed from Fig. 3.4(c). This is the reason, the performance of the DP is very close to that of CPU. However, our initial assumption for selecting the kernels was that the parallel kernel should perform well on GPU and badly on CPU, and the sequential kernel should perform well on CPU and badly on GPU. One assumption which is violated here is that the parallel kernel is performing relatively well on CPU. With this result, the further development of the model will be biased towards CPU.

Now, the next goal is to either keep searching for kernels that can satisfy the assumptions or idealize the values to fit the assumptions. The later one is chosen which makes the construction of the model simple and easy to understand. The values are reasonably assumed to create a powerful abstract model. Next section goes into the detail on constructing this model.

3.4. MODELING WITH IDEAL VALUES

An attempt was made to build the model using the real values in the last subsection. The speedup of the sequential kernel on CPU with respect to GPU range from 20 to 35 depending on the vector

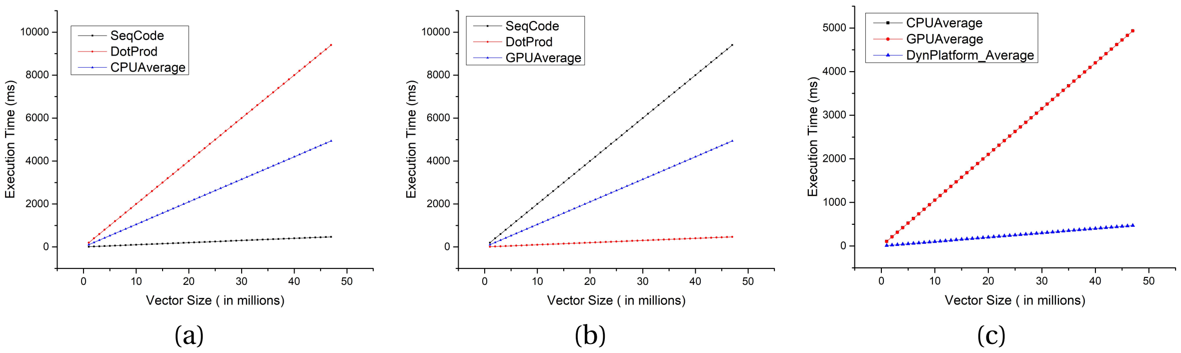


Figure 3.7: Idealized performance values of the sequential (SeqCode) and the parallel kernel (DotProd) kernels on (a)CPU, (b) GPU and (c) Dynamic Platform.

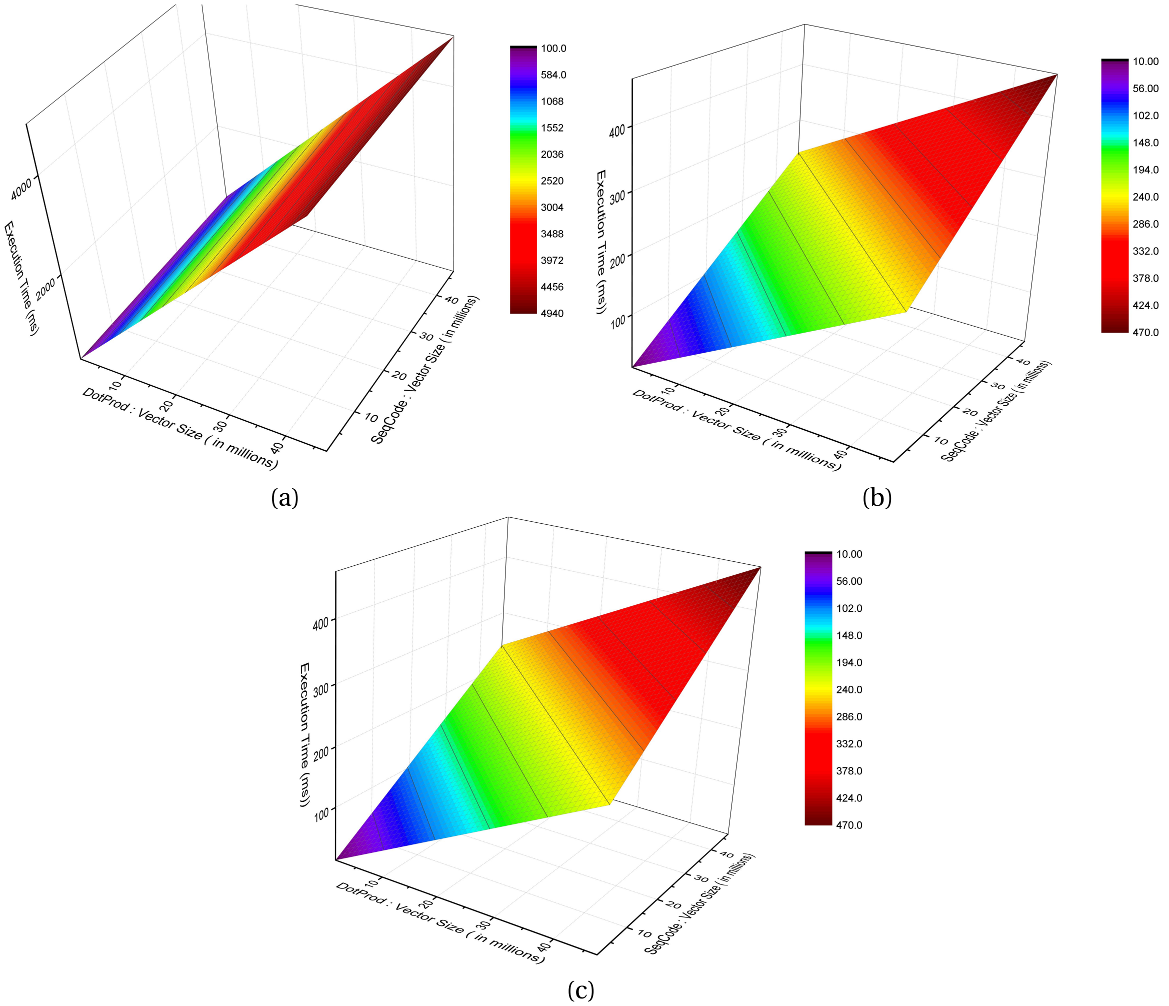


Figure 3.8: Surface plots depicting the average performances of the sequential kernel (SeqCode) and the parallel kernel (DotProd) for all combinations of vector sizes for (a)CPU, (b)GPU, and (c)Dynamic Platform. The values are idealized so that the average performance of both kernels for a same vector size is same across CPU and GPU.

size. On the other hand, the speedup of the parallel kernel on GPU with respect to CPU range from 10-20 based on the vector size. This makes the model biased towards CPU. In order to make the model independent of the platform, a common speedup of 20 is assumed for both the kernels. This means if the sequential kernel gets a speedup of 20 on CPU with respect to GPU then the parallel kernel would get the same speedup on GPU with respect to CPU. The resulting model will simplify our understanding and helps us to complete the development of the model.

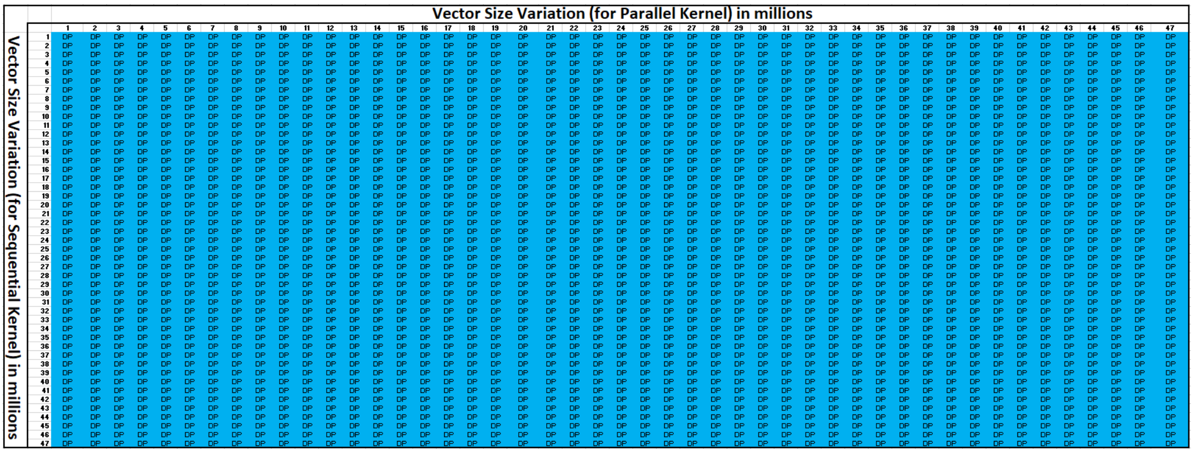
Fig. 3.7 shows the execution times for all the considered platforms. The values are assumed so that the average performance of both kernels is the same for both CPU and GPU. This is the reason the execution curves for Fig. 3.7(a) and Fig. 3.7(b) remain similar. This is more evident in Fig. 3.7(c), where the execution curve of CPU and GPU coincide. Now, there is a symmetry in the graphs which can be exploited to finish the model.

Fig. 3.8 depicts the surface plot values for all three platforms considered. Fig. 3.8(a) and Fig. 3.8(b) show that the CPU and GPU perform relatively well for sequential and parallel kernels respectively. This can be easily understood from the referred figures. For CPU, the performance plane in the direction of sequential kernel rise very slowly but linearly, and same is the case for GPU and the parallel kernel combination. For the other cases, the performance increase is quite linear. Fig. 3.8(c) shows the surface plot for the DP. The slope of the performance plane in both the direction of kernels is less when compared individually to either CPU or GPU. This is straightforward as the DP takes the best of CPU or GPU performances. This leads to the final part of the model in which the realistic scenario is embedded into the model to give important insights into the DP.

Fig. 3.9 shows the state of the DP up to this point in our model. The values in the x-direction correspond to the variation in the vector sizes for parallel kernel and the y-direction values belong to the vector size variation for the sequential kernel. The cells marked in blue indicate that the DP performs better than CPU/GPU. All the cells are marked blue in this referred figure as we assumed that the DP takes the best of both platforms (CPU/GPU).

In a realistic scenario, the frequency of the DP such as FPGA is less than that of either CPU or GPU by almost a factor of 10 or higher. This can be included in our model to understand how the performance of a DP varies when such a Reduction Factor (RF) is applied to it.

Fig. 3.10 clearly shows the effect of introducing the RF into our model and varying it from 1 till 13. Suppose, the RF of 3 is used in this model, this implies that all the values (averages) are divided by 3. Cells marked in green indicate the performance order from best to worst: GPU-DP-CPU. Similarly, the gray colored cells indicate the CPU-DP-GPU performance order. Red-colored cells indicate that DP is performing worse than CPU and GPU. When RF is 1, the DP performs well over the entire



Vector Size Variation (for Parallel Kernel) in millions

Vector Size Variation (for Sequential Kernel) in millions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
1	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
2	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
3	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
4	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
5	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
6	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
7	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
8	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
9	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
10	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
11	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
12	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
13	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
14	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
15	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
16	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
17	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
18	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
19	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
20	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
21	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
22	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
23	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
24	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
25	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
26	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
27	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
28	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
29	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
30	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
31	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
32	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
33	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
34	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
35	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
36	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
37	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
38	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
39	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
40	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
41	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
42	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
43	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
44	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
45	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
46	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP
47	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP	DP

Figure 3.9: State of the DP with no reduction factor applied. Blue colored cells indicate that the DP performs better than CPU/GPU for the vector sizes corresponding to that cell.

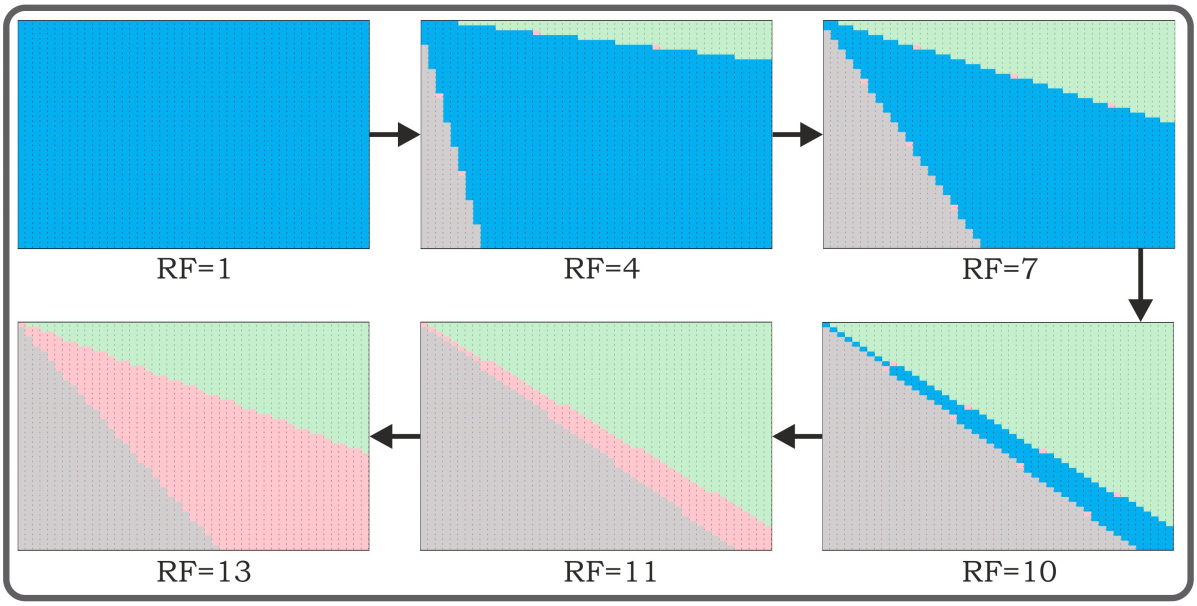


Figure 3.10: Behaviour of Dynamic platform with varying RFs. Performance order from best to worst for gray colored cells is CPU-DP-GPU and for green colored cells GPU-DP-CPU. Blue colored cells indicate DP is better than CPU/GPU and red colored cells indicate that the DP is worse than CPU/GPU.

region as assumed till this point. When the RF is increased from one, the interesting picture starts to appear. Towards the left and to the bottom area, corresponding to the lowest parallelism or highly sequential region, CPU starts dominating, and towards the right and to the top area belonging to the highest parallelism region, GPU starts dominating. This is evident from the fact that the CPU is designed to handle single threads effectively, and GPU for many threads. The area where the DP performs better than CPU/GPU (blue colored cells) shrinks towards the mid-region. This is the pivotal point of this model. The sweet spot, which lies in the mid-region corresponding to the medium level of granularity in parallelism shows the promising future of the DP. Even after decreasing the DP performance values by a magnitude less than those of CPU and GPU, it still performs better than these architectures. However, when the RF is increased beyond 10, the DP's performance starts becoming worse than that of CPU or GPU from the mid-region.

The main observations of the conceptual model can be summarized as follows:

- GPU is dominant in the top-right region where a high level of parallelism exists.
- CPU is dominant in the lower-left region where a low level of parallelism exists.
- DP performs better in the mid-region where the granularity in parallelism is neither too high nor too low.
- DP starts to perform worse than CPU and GPU when the RF is increased beyond the threshold (which is 10 here). The threshold is dependent on the assumed speedup. If the common speedup increases then the threshold also increases and vice versa.
- DP still stands as a second choice in the extreme ends (lower-left and top-right) of the spectrum even beyond the threshold point (RF = 10).

4

Implementation

Every science begins as philosophy and ends as art.
- Will Durant

This chapter gives detailed information on the proposed architecture implementation. The complete architecture consists of two distinct parts working for different application requirements. The Microblaze soft processor or the ρ-VEX dual-core platform caters to the sequential needs of the application, and the ρ-VEX manycore platform corresponds to the parallelizable applications. First, the design of ρ-VEX manycore platform is discussed in Section 4.1. Then, the Microblaze platform design is elaborated in Section 4.2 followed by the design of ρ-VEX dual-core platform in Section 4.3. Furthermore, the design of the full-system software needed to make the complete working of the dynamic system is explored in Section 4.4.

4.1. ρ-VEX Manycore Platform Design

The ρ-VEX manycore platform is a modified version of an original ρ-VEX streaming platform. It consists of one core per stream and each core operates independently of the other. This means that the application needs to be developed in such a way that if any communication is needed between two or more cores, then it has to be explicitly handled. This has advantages and disadvantages. It is advantageous because each core can complete its task quickly and efficiently without worrying for other cores' status. It is slightly disadvantageous in a sense that if at any point, the application needs synchronization for its threads, then it needs to be handled explicitly. This makes the platform very suitable for streaming applications like image processing.

4.1.1. Hardware Design

Fig. 4.1 shows the design and integration of ρ-VEX manycore softcore with Zynq Processing System (PS). The ρ-VEX manycore platform contains master and slave AXI buses to interface with other modules. Furthermore, each core can write to the DDR memory via an in-built DMA module. This is achieved via master AXI bus from the ρ-VEX core, which is connected to High Performance (HP) slave AXI bus of the Zynq PS. The application is downloaded directly to the instruction memories of all the ρ-VEX cores involved. This is done via master AXI bus from the Zynq PS to the slave AXI bus of the ρ-VEX manycore module. The results of an application can be accessed directly from the DDR memory existing on the PYNQ-Z1 board. All the resets are handled through PS Reset module which provides various reset signals such as peripheral asynchronous reset and interconnect asynchronous reset.

Fig. 4.2 reveals the configuration options available to the underlying ρ-VEX manycore platform. Number of streams, number of cores per stream, each core's instruction and data memories and

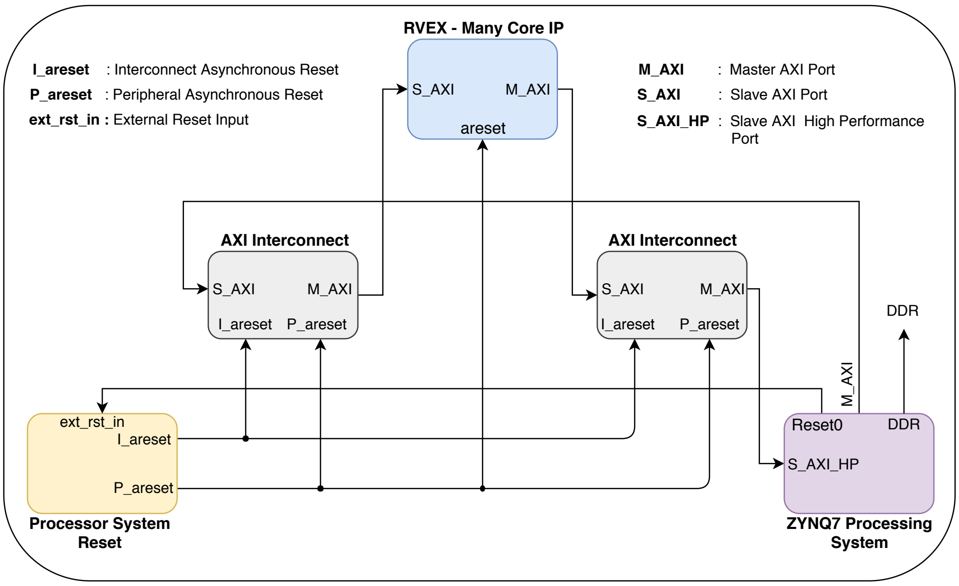
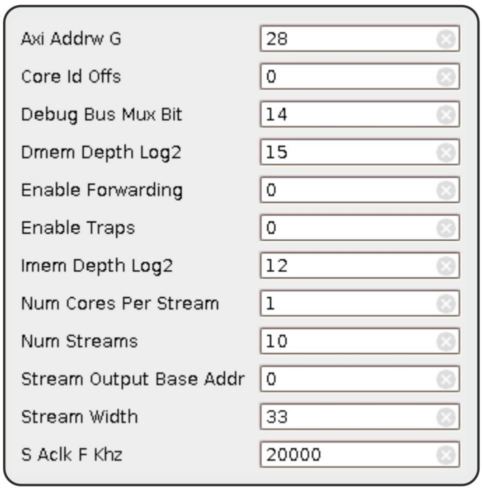


Figure 4.1: Main components involved in the design of ρ-VEX manycore platform

enabling forwarding are the main configurable options. For the softcore implemented, a ten stream with each stream containing a single core and each core containing a data memory of 32KB and instruction memory of 4KB is chosen. Forwarding is not enabled as it consumes more area and decreases the maximum frequency achievable. For this configuration, the frequency achieved is 120MHz. More cores cannot be synthesized as it exceeds the area available on the FPGA.



Axi Addrw G	28
Core Id Offs	0
Debug Bus Mux Bit	14
Dmem Depth Log2	15
Enable Forwarding	0
Enable Traps	0
Imem Depth Log2	12
Num Cores Per Stream	1
Num Streams	10
Stream Output Base Addr	0
Stream Width	33
S Aclk F Khz	20000

Figure 4.2: Available options to configure the manycore ρ-VEX platform. The options shown in this figure correspond to 10-stream ρ-VEX system with each stream containing one core and each core has an instruction memory of 4KB and a data memory of 32KB.

The resources consumed post implementation of the ρ-VEX manycore platform is displayed in Fig. 4.3. We have almost completely utilized all the resources available on a Zynq FPGA. Indeed, this is expected as the main goal of the manycore softcore is to utilize all the available resources to perform the computation.

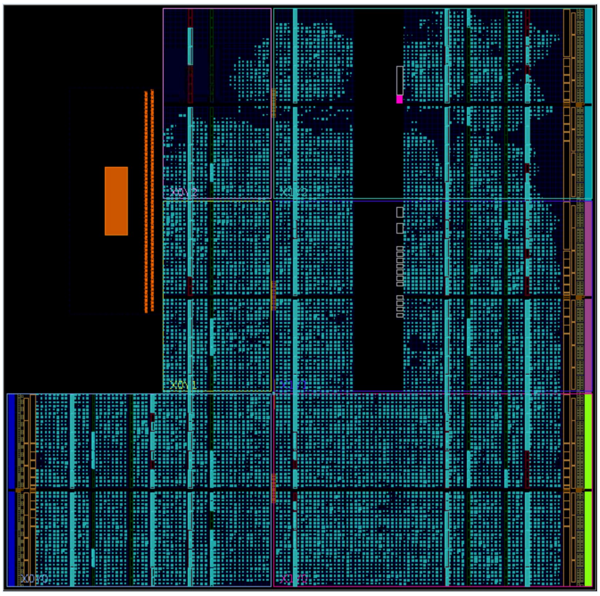


Figure 4.3: Figure is showing that the ZYNQ PL resources are almost completely exhausted for 10 streams, one core per stream RVEX platform, with each core having 32KB Data memory and 4KB Instruction memory.

4.1.2. Programming Toolchain

For compiling applications to run on the ρ-VEX cores, a special version of the ρ-VEX toolchain, also called mandelbrot streaming toolchain, is used. This toolchain exists for compiling the applications to 2-issue ρ-VEX core, and in addition, there exists no support for the division. It does support floating point operations via software emulation but at the cost of additional cycles as the required hardware does not exist on the implemented softcore. The ρ-VEX is big-endian whereas the ARM processor, existing on Zynq PS, is little-endian. In this regard, the data needs to be converted from one format to another if any communication occurs between an ARM processor and a ρ-VEX core.

4.2. Microblaze Soft Processor Design

Microblaze is a sophisticated softcore from Xilinx. In the context of our DP, this softcore caters to the sequential needs of the applications. Many of its parameters are configurable such as pipeline depth, size of instruction and data memories, the inclusion of MMU, etc. Additionally, it provides predefined configurations for maximum performance, minimum area, and maximum frequency. Maximum performance configuration is selected for our implementation. It also includes a branch prediction hardware to mitigate branch delay latencies.

4.2.1. Hardware Design

Fig. 4.4 demonstrates the important components in the design of Microblaze softcore to work in cohesion with Zynq PS. Microblaze communicates with DDR via master AXI port which connects to the HP-AXI port of Zynq PS. It communicates with a local memory via a specially designed Local Memory Bus (LMB) bus. It does not contain a slave AXI port which means, the ARM processor cannot communicate with Microblaze. To alleviate this problem, memory module of Microblaze is modified to give control to ARM processor. Fig. 4.5 shows the snippet of the modified Microblaze memory interface. Originally, the Microblaze consisted of separate instruction and data memories via dual port BRAM. Now the ARM needs access to this memory, and hence, one of the port is connected to the master AXI of Zynq PS via AXI BRAM Controller shown in a referred figure. This results in a unified instruction and data memories. The frequency of operation here is set at 100MHz above which the timing constraints are not met. The complete design is provided as a TCL script in

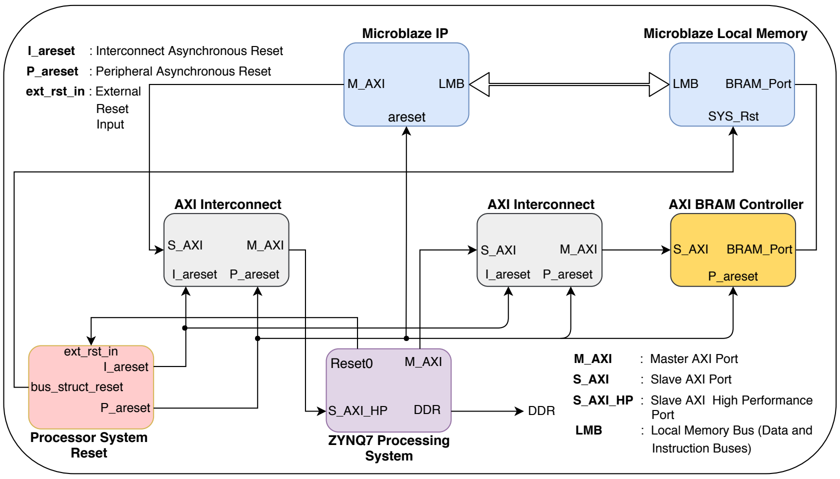


Figure 4.4: Essential components and their interconnections in the design of Microblaze single core processor

Appendix A, so that a reader can generate an entire architecture in a jiffy if needed.

Fig. 4.6 shows the area utilization by the Microblaze processor post-implementation. Microblaze synthesized consists of 128KB unified instruction and data memory which is the maximum supported by Xilinx. Even though the processor is made up of a lot of modules such as extended floating point unit, branch target cache for branch prediction, pattern comparator, integer divider, etc., it fills less than 1/5th of an FPGA area.

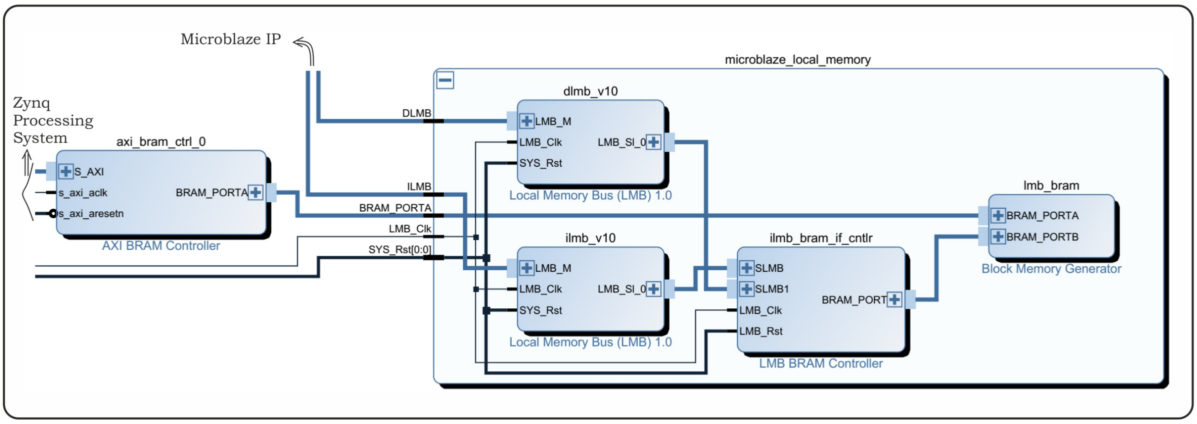


Figure 4.5: Design of the Microblaze Local Memory System interface. Dual port access is provided to the unified instruction and data memory in order to give control to both Microblaze and Zynq processing system.

4.2.2. PROGRAMMING TOOLCHAIN

Xilinx provides a GNU compiler collection (GCC) for Microblaze processor. The compiler tools supports both C and C++ programming languages via mb-gcc and mb-g++ commands respectively. Microblaze is little-endian based. Unlike ρ-VEX, this processor does not need to do any data conversion while communicating with an on-chip ARM processor, which is also little-endian.

4.3. ρ-VEX DUAL-CORE PLATFORM DESIGN

This section discusses the design of a dual-core ρ-VEX processor. As explained in the background chapter (2.4.1), this is an attractive softcore, which is huge and can utilize almost an entire FPGA area for doing computations. Each core is a 2-context, 4-issue processor containing a configurable L1 instruction and data caches. Further, L2 cache stocks in additional memory, which can provide

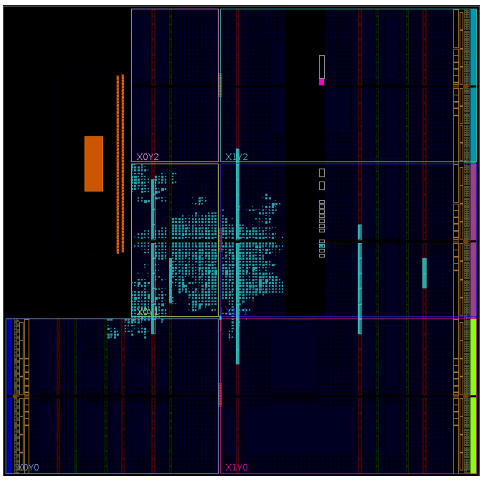


Figure 4.6: Resource utilization for single-core Microblaze processor with a maximum supported 128KB unified Instruction and Data memory.

coherence to both the cores and efficiently access instruction as well as data from the DDR memory.

4.3.1. HARDWARE DESIGN

Fig. 4.7 reveals the important aspects of the ρ-VEX and Zynq PS design. The ρ-VEX processor is connected to DDR memory via Accelerated Coherency Port (ACP) of Zynq PS. It is designed to operate at the 50MHz frequency. The ρ-VEX debug system is used to communicate with the ρ-VEX processor cores from the on-chip ARM processor. This debug bus operates via UART at much lower frequency (20MHz) so that it does not become a time-critical net. The dashed-line box inside the ρ-VEX toplevel module, called delay block/s, optionally exist/s between the L2 cache of the ρ-VEX processor and DDR memory. Each delay block provide a single-cycle delay to the data propagating from DDR memory to L2 cache. A number of these delay blocks are connected in sequence and synthesized, to understand the behavior of the ρ-VEX processor for varying distance of data path from the main memory to cache. This is further explored in the next chapter 5.2.3, which extensively discusses the

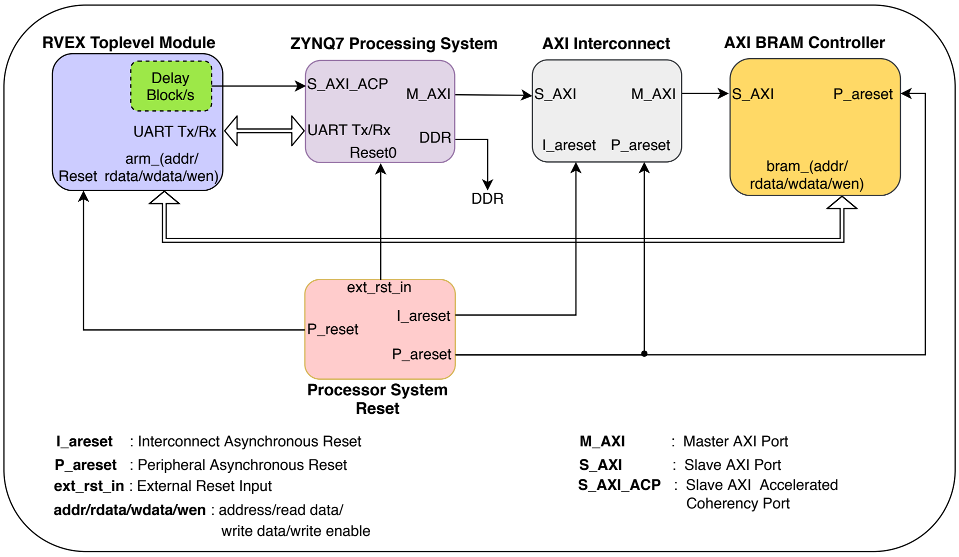
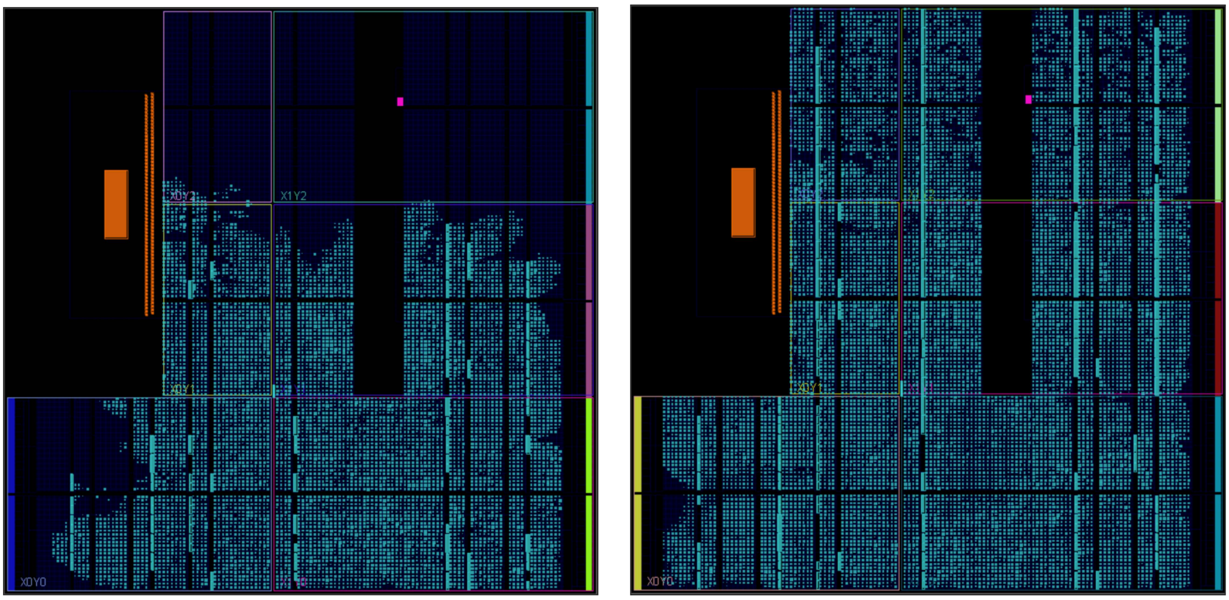


Figure 4.7: Design, depicting the important components, of the ρ-VEX dualcore platform. Delay block/s are inserted between the ρ-VEX L2 cache and the DDR port of Zynq Processing System.



(a) (b)

Figure 4.8: Resouce utilization for the ρ-VEX dual-core processor with (a) Minimal Cache Configuration (L2-128b, L1 Instruction Cache : 512b and L1 Data Cache : 128b) and (b) Maximum Cache Configuration (L2-256KB, L1 Instruction Cache : 4KB and Data Cache : 4KB).

evaluation of this platform among other ones. The on-chip ARM processor controls the ρ-VEX via AXI BRAM controller.

Initially, Zynq PS allocates a required portion of DDR memory (100MB here) to the ρ-VEX processor. In this regard, the application is dumped to this allocated DDR memory space, and the ρ-VEX processor starts executing it from there. Linux driver for allocating memory space was already developed in [39], and the same is used here.

Fig. 4.8(a) and Fig. 4.8(b) show the area consumption post implementation by the dual-core ρ-VEX processor with minimal cache and maximum cache configuration respectively. Unlike Microblaze, the cache sizes here are only limited by the space available on the FPGA. It is very clear from the referred figures that the dual-core ρ-VEX processor is capable of filling an entire FPGA area. If this transfers to a significant gain in performance is contemplated in the next chapter (5.2.3) but just to avoid curiosity, a substantial performance gain is observed from the results.

4.3.2. PROGRAMMING TOOLCHAIN

The standard ρ-VEX toolchain for a 4-issue core is used to compile the applications. Though the ρ-VEX is little-endian, no data conversion is required by an on-chip ARM processor to communicate with a ρ-VEX core as the ρ-VEX debug system handles the conversions internally. Floating point operations are software emulated as the required hardware is absent in the softcore.

4.4. SOFTWARE DESIGN FOR THE DYNAMIC PLATFORM

This section details the software design for the implemented DP. Fig. 4.9 shows a detailed outline of the flow of software design starting from reading the FPGA configuration bitstreams to reading back the final results. The software flow is elaborated as follows:

1. *Read the bitstreams*: This is the first step in the software flow in which the bitstreams are read from the Micro-SD card into the program memory. Two bitstreams are read. One corresponding to the parallel platform and the other to the sequential platform. Microblaze processor can

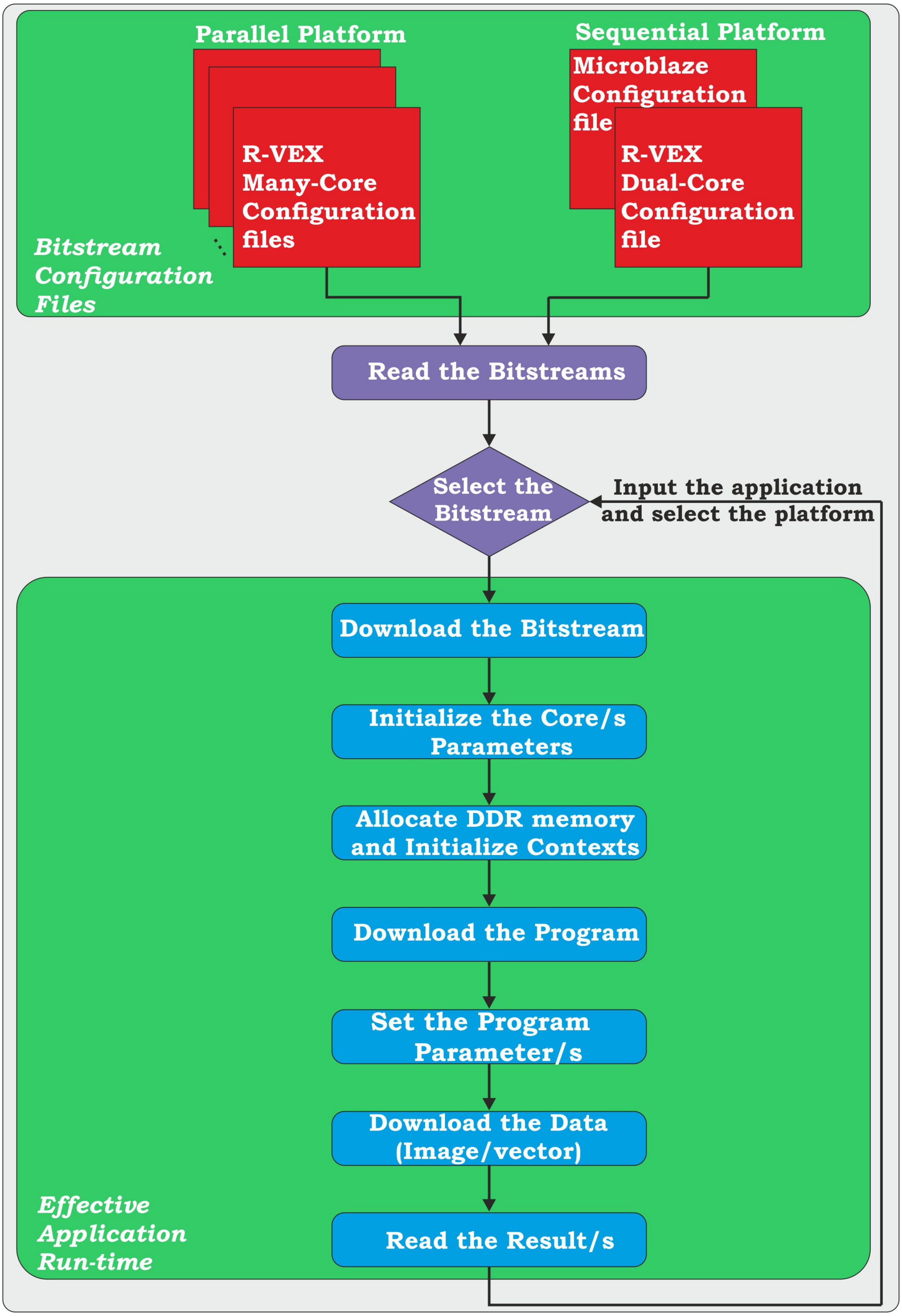

Parallel Platform
Sequential Platform
Microblaze Configuration file
R-VEX Many-Core Configuration files
R-VEX Dual-Core Configuration file
Bitstream Configuration Files
Read the Bitstreams
Select the Bitstream
Input the application and select the platform
Download the Bitstream
Initialize the Core/s Parameters
Allocate DDR memory and Initialize Contexts
Download the Program
Set the Program Parameter/s
Download the Data (Image/vector)
Read the Result/s
Effective Application Run-time

Figure 4.9: Complete flow of the DP software design

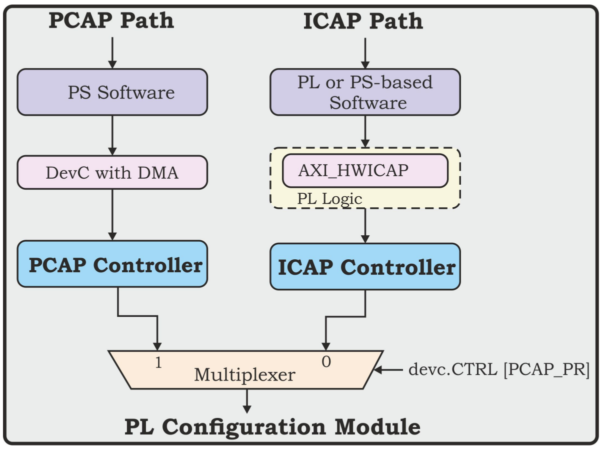


Figure 4.10: Available PCAP and ICAP interfaces for communicating with PL.

also be optionally loaded and used if the application under consideration is less than 128KB.

2. *Select the bitstream*: In this step, the program waits for the user inputs. The user selects the type of platform (parallel or sequential) to run the application on, and inputs the application.

3. *Download the bitstream*: The selected platform's bitstream is loaded from Zynq PS to PL in this step. The downloading paths are shown in Fig. 4.10. Processor Configuration Access Port (PCAP) is used here and this is the most commonly used path. Device Configuration interface (DevC) is accessed using the Xilinx provided linux driver (xdevcfg). To use Internal Configuration Access Port (ICAP), one needs to instantiate AXI_HWICAP module [47] in the PL before using this path and change the default select value of the multiplexer.

4. *Initializations*: Various core related initializations are performed in this step. The important part here is to map a certain part of the ARM processor memory to the cores of ρ-VEX manycore or Microblaze so that any crucial information can be communicated easily. This can be the details on the image partition or some flags to know if the data is processed. This step is not applicable to the ρ-VEX dual-core processor as it does not have a slave AXI bus to communicate, instead the communication happens via the ρ-VEX debug system (UART).

5. *Memory allocation*: Python memory server is used to allocate a portion of DDR memory to ρ-VEX manycore or Microblaze processor so that they can directly write the results in the DDR memory. For the ρ-VEX dual-core processor, a linux driver is used to allocate DDR memory.

6. *Program download*: In this step, the program is directly downloaded to the instruction memories of the cores in case of ρ-VEX manycore and Microblaze processor. However, in case of the ρ-VEX dual-core processor, the program is downloaded to the DDR memory from where it starts executing.

7. *Setup of program parameters*: The program parameters are set in this step. The memory mapped in step 3 is used here to communicate with the cores. Program parameters depend on the type of application under consideration. For image processing, the parameters are image segregation details and the address of DDR memory to which the results are written to.

8. *Download data (if any)*: In this step, the application data (image/vector) is downloaded chunk by chunk directly into the data memories of the cores. Data is divided depending on the size of the data memory and is consumed in sequence.

9. *Read the results*: This is the last step in which the results of the application are read. As the processors directly write the results to the DDR memory, they are in turn accessed from this location and interpreted. For instance, the image is reconstructed in case of image processing application. The code goes to step two and repeats again ensuring the dynamic capability of the platform.

The detailed instructions along with the materials used and developed in this thesis is made open source and is available on [48]. Individual softcores as well as the fully functional dynamic platforms which can switch between two (ρ-VEX dual-core and ρ-VEX manycore) or three (ρ-VEX dual-core, ρ-VEX manycore and Microblaze) different architectures are provided so that it is easier to replicate the work done and use/improve the functionality further.

5

Experiments, Results and Analysis

Everything is theoretically impossible, until it is done.
- Robert A. Heinlein

This chapter centers on the detailed discussion of the experiments and results. First, the benchmarks used are described in Section 5.1. Then, the three platforms used in this thesis are evaluated against these benchmarks in Section 5.2. Further, the DP analysis is carried out similar to the one we did in the concept modeling chapter in Section 5.3. Section 5.4 dwells on the implications of the DP on overall execution time of the application. The ρ-VEX softcore performance is compared with the existing ARM hardcore in Section 5.5.

5.1. Benchmarks for the Platform Analysis

This section describes the three benchmarks that are used in the evaluation of all the implemented softcores.

5.1.1. Image Processing Benchmark

This benchmark is used to test the amount of parallelism exhibited by the underlying softcore. Three operations are done on an image in this benchmark. Fig. 5.1 shows the main steps done on an input image. First, the original color image is gray-scaled. Then the image is blurred using gaussian blur, typically to reduce image noise and reduce detail. Finally, the Sobel filter is applied to detect all the edges present in the original image.

5.1.2. Hash Benchmark

This benchmark is used to test the use of huge caches which is one of the key characteristics of the sequential platform. The hash function used here is one of the best string hash function, called djb2,

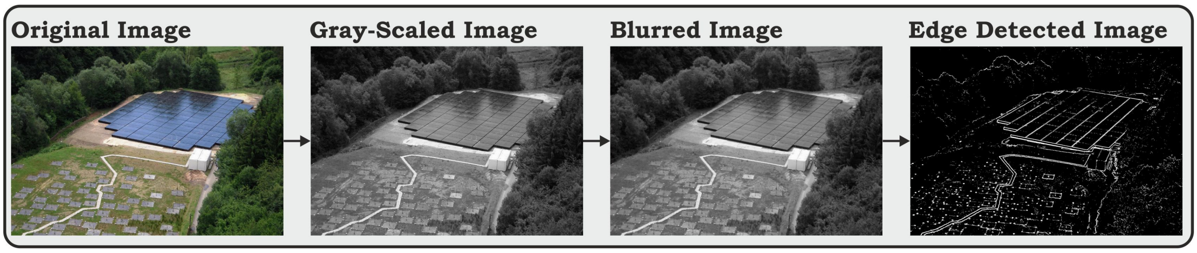


Figure 5.1: Image processing benchmark: First, the image is grayscaled, then blurred using gaussian blur and finally edges are detected using a sobel filter.

which was reported by Daniel Bernstein [49] and has excellent distribution and speed on many different sets of keys and table sizes. Initially, a program is written to extract as many unique words as possible from an English dictionary using the selected string hash function. Now, the benchmark utilizes either a part or complete strings (depending on the hash table size) that are extracted and finds the keys of these strings. The keys are stored in sorted order in an array, also called a *hash table*. Binary search is performed to find the particular key for a random string. Traditional collision techniques are not followed as it complicates the evaluation of softcores by focusing more on dynamic allocation of memory. Instead, the unique keys (0-262143) are calculated and stored in a hash table of desired size (much less than the maximum value of the unique key, which is 262143 here). For example, if 20000 entries are required then an integer array is allocated with a storage capability of 20000 items and each item can have any value between 0 and 262143. The value 262143 (2^{18}-1) is selected to convert a heavy modulus operation to a simple AND operation (i.e., m % 2^n = m & (2^n-1), which is used extensively in the hash function.

5.1.3. CRC32 Benchmark

CRC32 benchmark, or simply a CRC benchmark, is used to test the sequential performance of a softcore. It is a part of MiBench [50] which performs a 32-bit Cyclic Redundancy Check (CRC) on a file. CRC checks are used quite often to detect and correct errors in data transmission. The sound files from the ADPCM benchmark [51] serves as the data input.

5.2. Platforms Evaluation

In this section, the evaluation of the developed platforms is done with respect to the benchmarks described in the previous section. The main code, which acts as a scheduler, resides in the Zynq PS. The on-chip ARM processor executes this code, downloads the bitstream configuration file, binary application file, image/vector data (optionally), and finally reads the output (image/vector) from the DDR memory as previously described in 4.4.

5.2.1. ρ-VEX Manycore Platform Evaluation

The ρ-VEX manycore platform, which is configured here to evaluate, consists of 10 streams. Each stream contains a core which has 32KB data memory and 4KB instruction memory. Hence, the streaming platform is converted to a manycore platform by cutting down cores in each stream to just one. The frequency of operation is 120MHz. Image processing benchmark, which exhibits ample parallelism to exploit, is run on this platform. The function for dividing image based on the

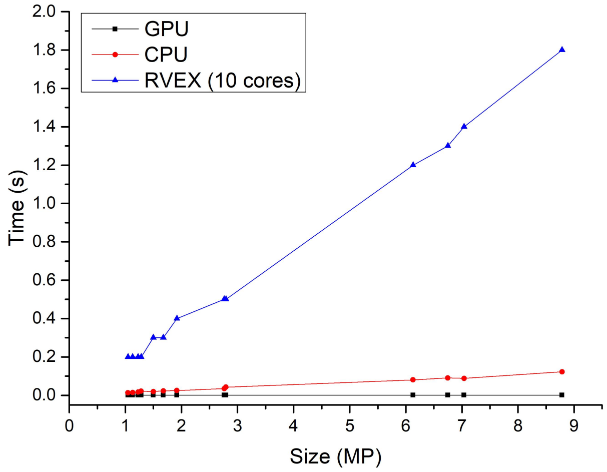


Figure 5.2: Evaluation of ρ-VEX platform for Image processing Benchmark

number of cores and the amount of memory available is taken from [52].

Fig. 5.2 shows the performance of the ρ-VEX with respect to CPU and GPU platforms. The CPU and GPU platform details are same as mentioned previously in Table 3.1. The size of the image in MegaPixels (MP) is shown along the x-axis and the computation time in seconds (s) is marked along the y-axis. GPU time usage is very less compared to the other two platforms and remains almost constant for the sizes of image considered here. This is evident from the fact that it contains 1000s of cores with millions of threads operating together. CPU time taken is one order magnitude less than the ρ-VEX platform. The increase in slope is very slow but notable when compared to GPU. The ρ-VEX manycore platform, expectedly, takes more time to complete the task. The slope, which is around 0.26 here, seems to be very steep when compared to that of CPU or GPU. The results are in line with our expectation. As the ρ-VEX platform has significantly lesser resources and runs at a magnitude or two lower than CPU or GPU frequency, the results can be comprehended with ease.

Core-To-Core Comparison: To put things into context, for evaluating the ρ-VEX platform, it is compared core to core against CPU. The CPU, considered here, is the same as the one used to develop concept modeling with its architecture shown in Fig. 3.3. It consists of 2 powerful cores capable of executing at least 6 instructions per cycle per core running at 2300MHz. Now, the question is how many ρ-VEX cores are required to match this CPU. This can be easily calculated as follows.

$$\begin{aligned}\begin{array}{c}\text{Number of } \rho\text{-VEX cores required}\\ \text{to match CPU}\end{array} &= \frac{\text{CPU frequency*Number of Cores * Instructions per cycle}}{\rho\text{-VEX frequency * Issue width}}\\ &= \frac{2300 * 2 * 6}{120 * 2} = 115\end{aligned}$$

In order to match the mighty CPU, 115 ρ-VEX cores are required. Currently, we just have 10 cores. Now, normalizing the CPU values by 11.5 (115/10), we are ready to compare CPU and ρ-VEX core to core. Fig. 5.3 depicts the required core-to-core comparison of CPU and ρ-VEX processors. It is very clear from the referred figure that both CPU and ρ-VEX core perform very closely. As this benchmark does not utilize the complex CPU super-scalar architecture other than the big caches, it can be fairly argued in this scenario that the CPU frequency and its multiple execution units are the dominating factors contributing to its performance.

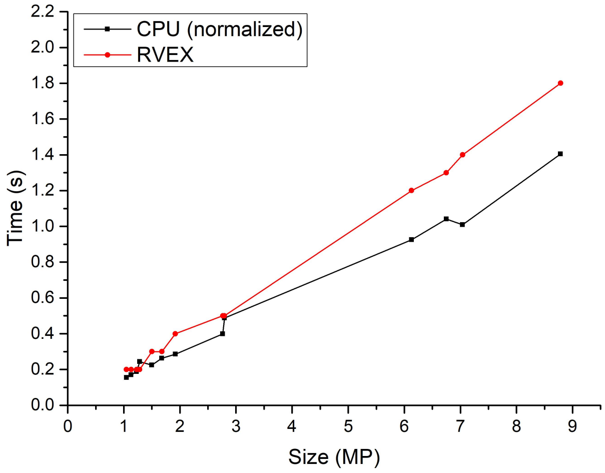


Figure 5.3: Core-to-Core comparison of ρ-VEX and CPU

5.2.2. MICROBLAZE PLATFORM EVALUATION

Microblaze is a single core processor operating at 100MHz. It is built to handle sequential applications. Due to the restriction on the size of local memory from Xilinx, only 128KB unified instruction

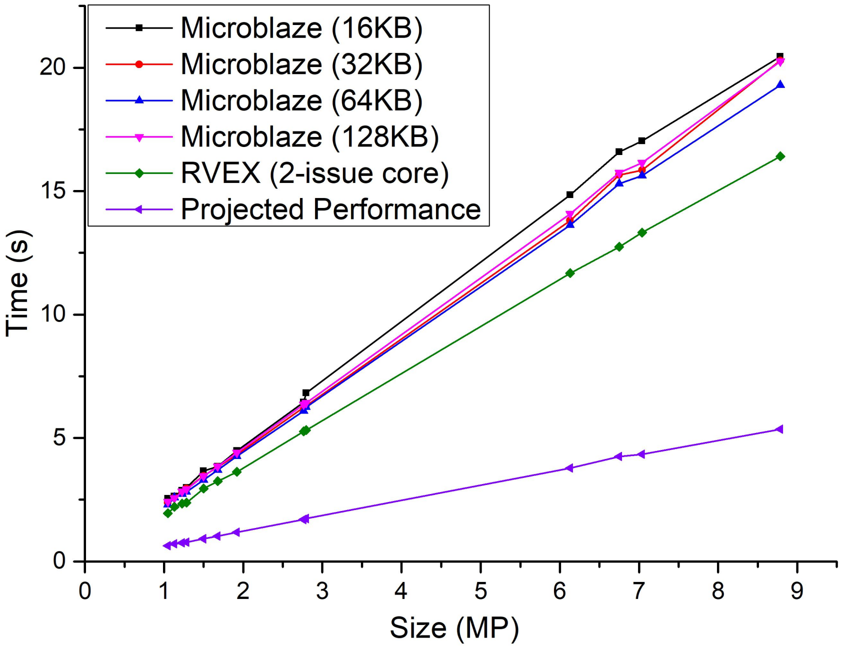


Figure 5.4: Evaluation of Microblaze Platform for Image Processing Benchmark

and data memory is implemented. It does provide some branch target cache which is used in the branch prediction mechanism. The high-performance configuration, which is a pre-defined configuration, is selected which implements a 5-stage pipeline. It is evaluated with image processing benchmark as well as CRC benchmark.

Fig. 5.4 shows the performance of Microblaze processor for various memory configurations with respect to image processing benchmark. There is a very little variation among different memory configurations of Microblaze. This is not to be confused with a cache memory. All the code already resides in the internal memory (BRAM). Only image-pixel data is brought-in chunk by chunk until all the data is processed. As the ARM processor utilizes its caches (512KB L2 cache) efficiently to transfer image data via AXI bus to Microblaze's internal memory, the size of the internal memory is not quite important once all the code resides inside it. Effect of internal memory may be observed if it exceeds 512KB in which case the performance is not yet saturated. The ρ-VEX curve for a single core is also mentioned to get an idea on the comparison. The ρ-VEX core is 2-issue wide and runs at 120Mhz whereas Microblaze is a single issue and runs at 100MHz. The difference in speedup is expected to be about two, but it is less than that here (1.3) as all the ρ-VEX bundles are not filled completely. There is an extra curve, called projected performance, which is modeled by increasing the Microblaze frequency to 360MHz which is easily achieved in Virtex-7 devices (The frequency can go till 560MHz for Zynq Ultrascale+ device which is a high-end version of the Zynq FPGA). The curve shows that the Microblaze can perform much better than its current implementation and its frequency is currently limited by the Zynq FPGA.

Fig. 5.5 shows the behavior of Microblaze processor towards CRC benchmark. There exists no parallelism in this benchmark which makes it suitable for a single core processor like Microblaze. Different sizes of CRC in Million Characters (MC) are simulated on four different memory configurations of Microblaze processor and CPU. Two observations can be made here. First, as discussed in the previous paragraph, and proving now for sequential application, there is no significant performance difference for varying memory sizes. Second, the CPU performs much better, and the difference in performance with Microblaze processor is analyzed further.

Core-To-Core Comparison: Microblaze is compared head-on with CPU on the core-to-core basis, as we did previously for the ρ-VEX processor. In the previous analysis, we considered not just

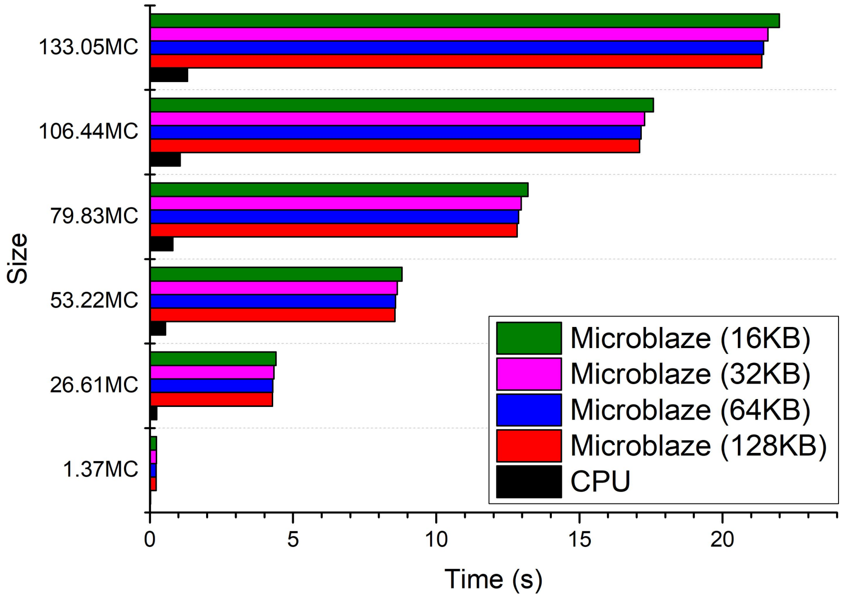


Figure 5.5: Evaluation of Microblaze Platform for CRC benchmark. MC: Million Characters

the difference in the frequency but also the number of execution units available to simultaneously run multiple threads. However, in this section only frequency difference is considered as we are dealing with sequential application and is assumed to run on a single thread. CPU runs at 2300MHz and Microblaze at 100MHz which gives the difference factor of 23 in performance directly. Scaling the CPU values by 23, we get the performance curves as shown in Fig. 5.6. Just by considering a difference in frequency, Microblaze and CPU performances are very close. Even better, the Microblaze performs slightly better than CPU in this normalized case. This gives a very good hope in the future of reconfigurable architectures as the frequency of their operation gets comparable to that of mainstream CPUs.

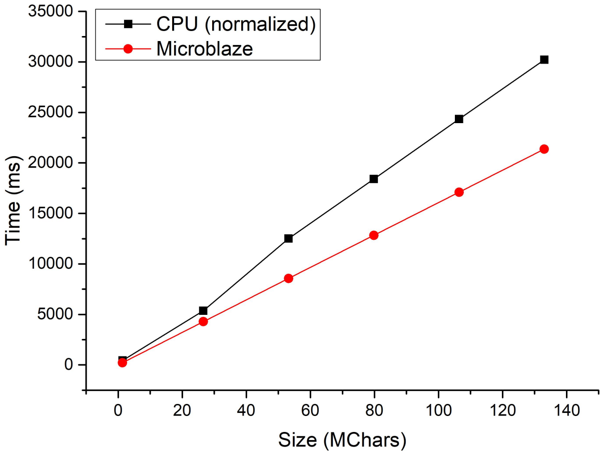


Figure 5.6: Core-to-Core comparision of Microblaze vs CPU for CRC benchmark

5.2.3. ρ-VEX Dual-Core Platform Evaluation

In this subsection, an evaluation of ρ-VEX dual-core Platform is done extensively. The ρ-VEX platform implemented consists of two cores, each capable of issuing 4 instructions simultaneously, and each core contains an L1 instruction and data caches of 4KB and a shared L2 cache of up to 256KB. It operates at 50MHz and is capable of executing instructions directly from DDR. A 100MB of space

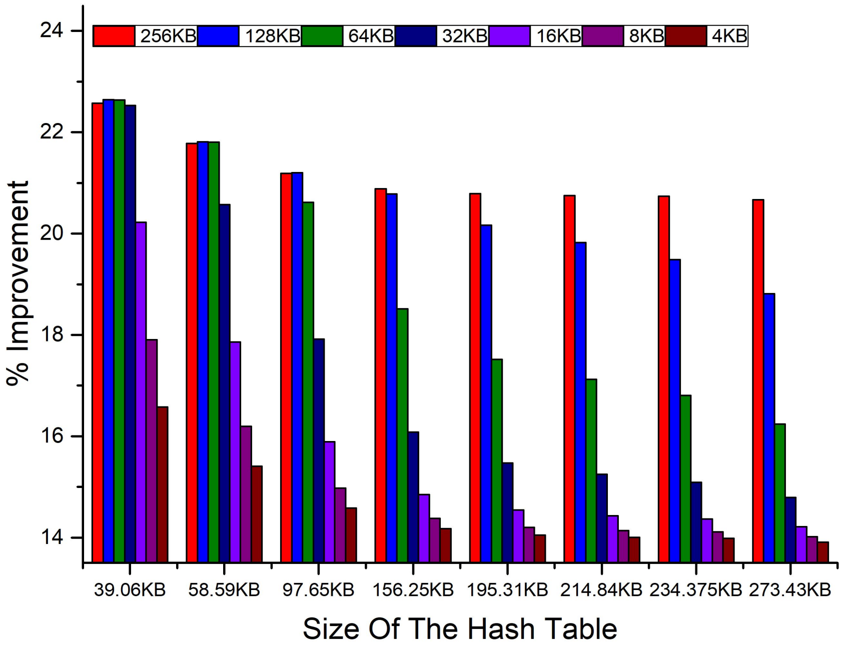


Figure 5.7: ρ-VEX-Dual core evaluation with respect to Hash Benchmark. All the percentage improvements are done with respect to minimal cache configuration of 768b (L2-128b, L1 Instruction Cache: 512b and L1 Data Cache: 128b)

is allocated in DDR memory (512MB in total) exclusively to the ρ-VEX platform. The scheduler running on the on-chip ARM processor downloads the application binary to the allocated DDR space and communicates with the ρ-VEX processor using the dedicated ρ-VEX debug system. The effect of cache size and the number of delay cycles from DDR memory to L2 cache on the applications are discussed next.

CACHE EFFECT ON PERFORMANCE

Fig. 5.7 shows the behavior of the ρ-VEX processor to the hash benchmark. The size of the hash table used is mentioned along the x-axis and the percentage improvement in the execution time of the application for various L2 cache sizes are plotted along the y-axis. The performance improvement is measured with respect to minimum cache size configuration which consists of 128b L2 cache, 512b of L1 Instruction Cache and 128b L1 Data Cache, totaling 768b. Fully disabling a cache system is not possible in the current softcore framework, which is why all the references are made with respect to minimal cache size configuration. Just to give an idea on the runtime, from which the plot of percentage improvement vs. hash table size (depicted in Fig. 5.7), is created, the execution time values are shown in Table 5.1. The important observations from the Fig. 5.7 are as follows:

Table 5.1: Execution times for the ρ-VEX processor of different cache configurations with respect to various hash table sizes of the hash benchmark.

Size of the	**Runtime in seconds for different ρ-VEX L2 cache sizes**							
Hash Table	**256KB**	**128KB**	**64KB**	**32KB**	**16KB**	**8KB**	**4KB**	**768b**
39.06KB	7.01	7.00	7.00	7.01	7.22	7.43	7.55	9.05
58.59KB	15.82	15.81	15.81	16.06	16.61	16.95	17.11	20.22
97.65KB	43.82	43.81	44.14	45.64	46.76	47.27	47.49	55.60
156.25KB	112.39	112.54	115.76	119.22	120.97	121.64	121.93	142.06
195.31KB	174.92	176.29	182.14	186.66	188.71	189.46	189.80	220.82
214.84KB	212.11	214.59	221.81	226.83	229.02	229.79	230.17	267.64
234.37KB	252.40	256.39	264.93	270.40	272.70	273.51	273.91	318.44
273.43KB	344.53	352.60	363.76	370.06	372.57	373.43	373.88	434.29

- Up to 23% improvement in the performance is observed by increasing the L2 cache size till 256KB.

- Once the size of the L2 cache can accommodate all the entries of the hash table, then the performance is nearly saturated for other higher size caches as there is no more cache misses. For example, consider the hash table size of 39.06KB. The performance is increasing until the cache size of 64KB which is greater than 39.06KB and saturates after this size. To illustrate one more example for better understanding, consider hash table on an extreme end which is 273.43KB. Now, this size is greater than the maximum possible cache on the current FPGA which is 256KB. Now the performance never saturates, and it is clear from the picture that every cache size configuration performs better than its smaller versions.

- The difference between the percentage improvements from one cache configuration to the other increases with the size of the hash table. Consider the hash table sizes of 195.31KB and 273.43KB, for instance. The difference in performance improvement between 256KB and 64KB cache configuration is 3.27% for a hash table size of 195.31KB. This difference increases to 4.43% for 273KB hash table size. Similarly, it is clearly visible from the referred figure that the performance improvement difference for any two different cache configurations increases with increase in hash table size. The reason for this increase in difference is that there are more misses experienced by the smaller size cache configuration with increase in hash table size than the higher size cache configuration.

- Maximum achievable percentage improvement decreases with increase in hash table size for a cache size that can fit in all the entries of the hash table, stabilizing at some point. This is true here only for 256KB cache configuration which has the highest of 22.5% improvement but stabilizes at 20.6%. Other cache sizes do not follow this pattern as they cannot fit in all the entries and hence, the performance is not yet saturated. The reason is simple. For smaller runtimes, the percentage increase seems bigger. Let's consider the difference between 7.5s and 15s runtime with the difference between 75s and 100s runtime. The former case gives

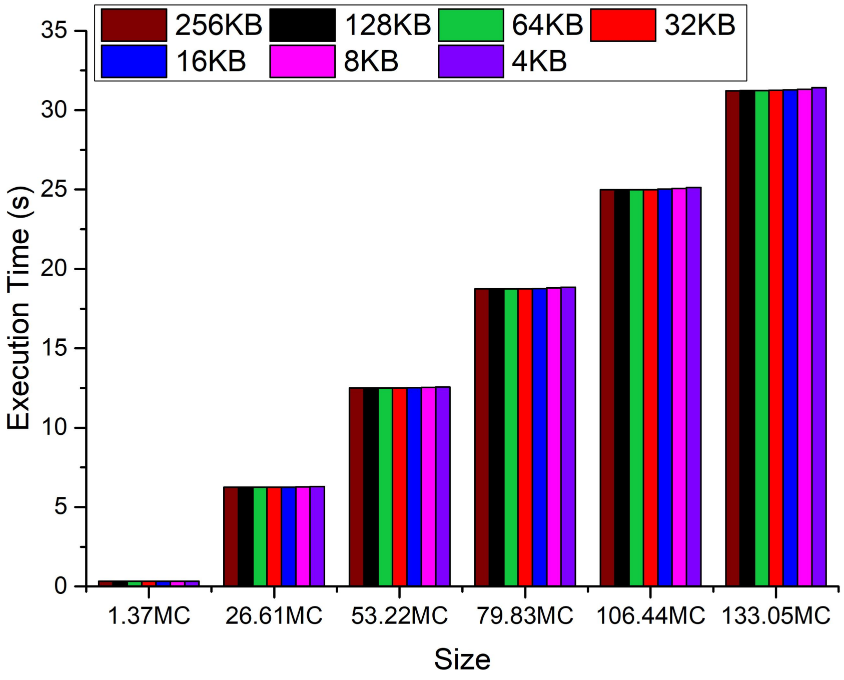


Figure 5.8: Evaluation of ρ-VEX dual-core with respect to CRC benchmark

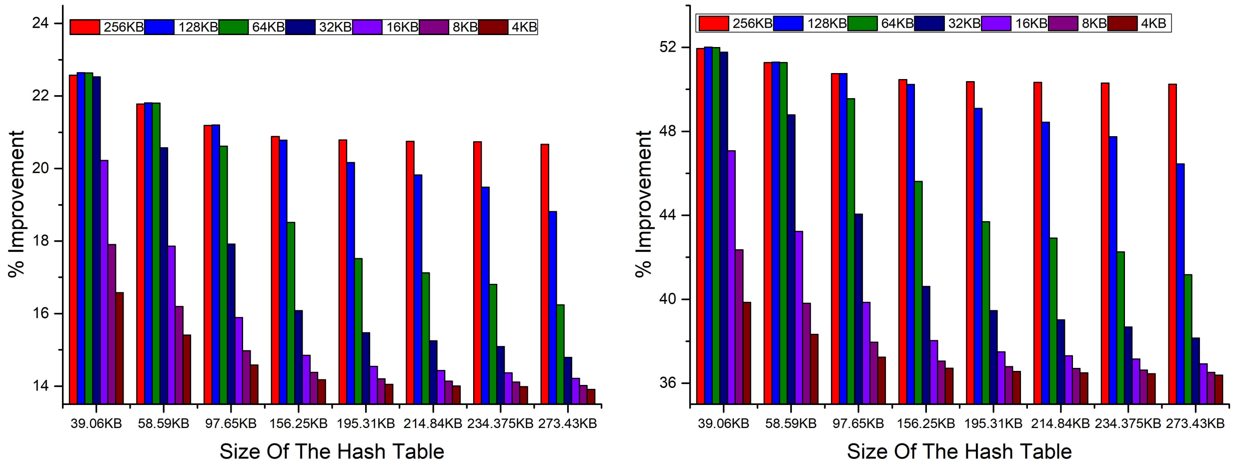
50% improvement for a difference of 5s runtime while the latter case gives 25% improvement for 25s runtime difference. Using this fact and observing the Fig. 5.7, we can understand that the maximum percentage performance improvement is stabilized once the runtime is significant enough.

- Bigger caches are the essence of the modern CPU architectures and can be considered as one of the important dimensions of a sequential application. The effect of having these huge caches is more clear than ever with this application.

Let us now consider the CRC benchmark, which incidentally does not take advantage of the huge caches even though it is a sequential application. Fig. 5.8 depicts the performance of the ρ-VEX platform for this benchmark. The CRC benchmark consists of a small feedback table of 256, 32-bit entries. Each data file that needs CRC check utilizes this table for every word (4 bytes) present in it to calculate and update the CRC value. As is evident from the referred figure, the execution time hardly varies with an increase in the size of the cache. Once the application fits in the cache, the performance saturates for the underlying platform. The application binary in this case, containing a feedback table, fits in the 4KB cache and any further processing of data needs no more than an entry, at a time, from the huge data file residing in the DDR memory. As each data is processed only *once* from start to end in an ordered manner, the cache accesses are perfectly sequential, and this means, a bigger cache does not translate it to performance. This proves that having huge caches is not significant to all sequential applications. Other dimensions of the sequential applications such as branch prediction, out of order or speculative execution may be essential in order to improve their performance.

Delay Based Parametric Performance Analysis

In the previous discussion, we observed that for the hash benchmark up to 23% performance improvement was achieved. If we take a step back and analyze the number of cycles taken for bringing a data from DDR memory to the L2 cache, we get a different picture. In our current architectural framework of the ρ-VEX on the Pynq-Z1 board, it takes just *15* cycles to get a piece of data to L2 cache from DDR memory. However, in reality, memories are situated far away from the main processor. Typical FPGAs that are becoming increasingly important for data centers, take around 150-200 cycles to get a data from an external peripheral to the processor's cache memory. This scenario



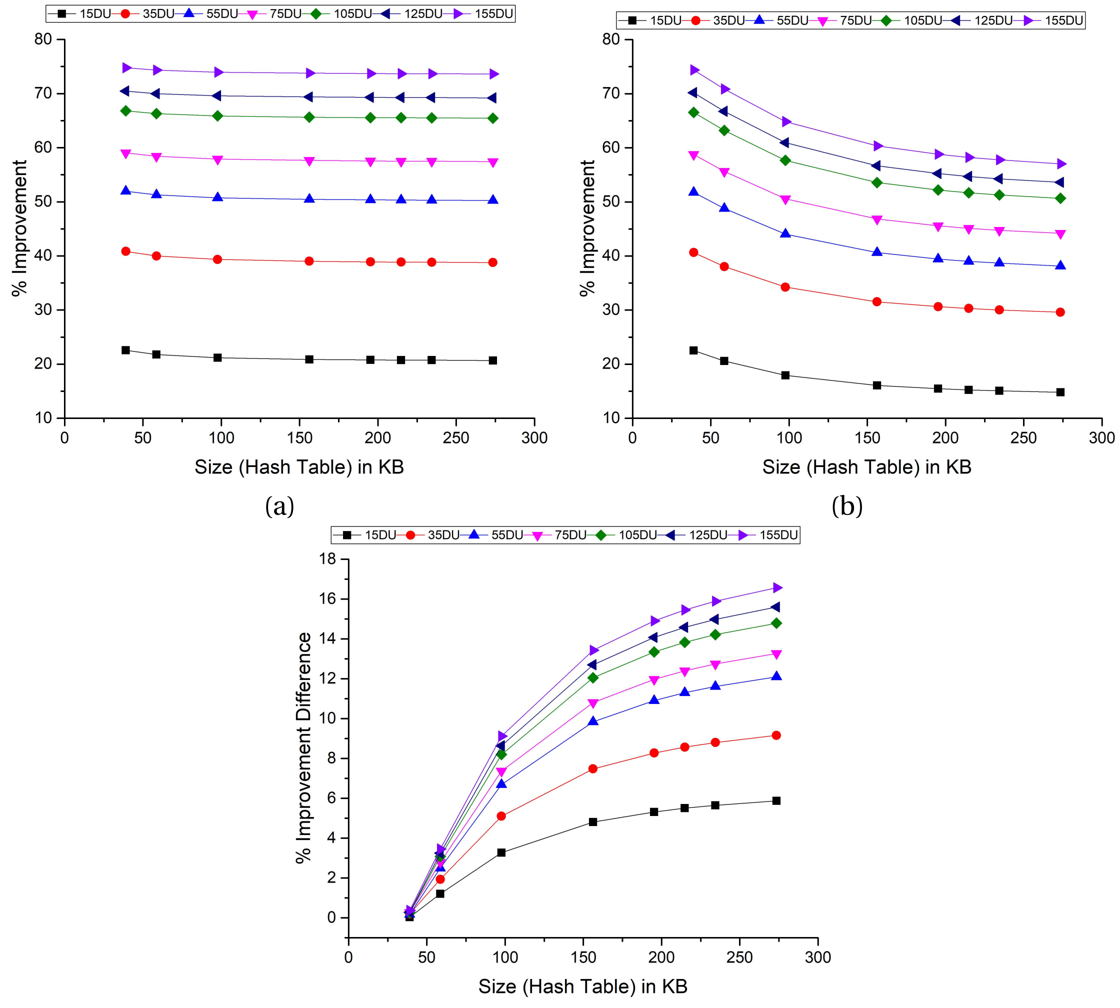
(a) (b)

Figure 5.9: % Performance improvement for the ρ-VEX processor for different cache configurations with respect to minimal cache size (768b) and having (a)15 delay cycles (original) and (b) 55 delay cycles (40 added DUs) between DDR and L2 cache.

can be simulated in the ρ-VEX processor by inserting a chain of delay elements, each providing a single cycle delay, in the path between L2 cache and DDR memory.

Fig. 5.9 shows the effect of introducing the delay elements on the performance improvement percentages of the hash benchmark. Fig. 5.9(a), same as Fig. 5.7, is shown here as a reference to compare with the ρ-VEX platform consisting of a newly added 40 Delay-Units (DUs), which is shown in Fig. 5.9(b). The first observation is that the highest percentage improvement is increased from 23% to about 52%. Let us compare the difference in percentages for both the figures with cache sizes of 256KB and 64KB corresponding to the hash table size of 273.43KB. The original ρ-VEX platform has a difference of 4.43% whereas the ρ-VEX platform with 40 DUs has 9.08% difference. This can be easily fathomed. Now, each access to data from the DDR memory is increased by 40 cycles. Hence, the cache miss penalty is relatively high in this case. Higher cache sizes have fewer misses, and so they experience less penalty in time when compared to smaller cache sizes. Indeed, we have amplified the cache miss time which is what contributes to the increase in the difference. This is true for any two different cache sizes, which can be easily observed by comparing the values in Fig. 5.9(a) to the equivalent values in Fig. 5.9(b).

Next, we compare how the change in delay affects the ρ-VEX dual-core platform (256KB mem-



(a)

(b)

(c)

Figure 5.10: % Performance improvement for (a)The ρ-VEX dual-core platform (256KB memory) and (b) The ρ-VEX manycore platform (32KB memory) with respect to the hash benchmark. (c) shows the difference in percentage performance improvement between these platforms at a corresponding hash table size.

ory) and the ρ-VEX manycore platform (32KB memory). For the purposes of this experiment, a core of the ρ-VEX dual-core platform with 32KB cache size is assumed as a core belonging to a ρ-VEX manycore platform. This assumption is made as the ρ-VEX manycore is incapable of executing large programs, that doesn't fit in the instruction memory, from DDR memory. It is a reasonable assumption as only one core of the manycore platform would be used for the hash benchmark for dependency (sequential program) reasons and both the architectures are ρ-VEX, and the memory capabilities are made same as each core of the manycore ρ-VEX platform has 32KB internal memory. This analysis is called delay based parametric performance analysis.

Fig. 5.10 shows the behavior of both platforms for a range (15-155) of delay cycles with respect to hash benchmark. Fig. 5.10(a) reveals the performance of the ρ-VEX dual-core with a highest L2 cache size of 256KB. A whopping 75% highest percentage improvement is observed for the ρ-VEX processor with 155DUs. Secondly, the point mentioned in the last subsection is relevant here. It was explained that the maximum percentage improvement is stabilized once the runtime is significant enough and including the fact that all the hash entries can be fit in the cache. This is clearly visible in the referred figure. For example, consider the curve with 155DUs. For the hash table size of 39.06KB, the percentage improvement is 74.77 but it stabilizes at nearly 73.66% for hash table sizes greater than 150KB.

Fig. 5.10(b) paints a different picture for the ρ-VEX manycore platform. It can fit only 32KB hash table as the internal memory is limited to 32KB. This is the reason the curves show a downward trend. Initially, for a 39.06KB hash table, the percentage improvement is highest as it can fit in nearly all entries of the hash table but for higher sizes of a hash table, the percentage improvement decreases as the cash miss rate increases.

Fig. 5.10(c) shows the difference in percentage improvements for both the platforms at respective sizes of hash table. Initially, the difference is almost 0 as both the platforms can accommodate nearly all the hash entries, but once the size of the hash table grows, the %-improvement difference also grows with it as the performance of the manycore ρ-VEX is not yet saturated. Around 17.5% difference in improvement is observed for hash table sizes more than 200KB. This is a significant difference, and how significant this difference constitutes to our DP is explored in the next discussion.

5.3. Dynamic Platform Analysis

In this section, the analysis of the DP is done using the results of the platform evaluation. The term 'DP' refers to the combination of the ρ-VEX dual-core platform (256KB memory), for sequential applications like the hash benchmark, and the ρ-VEX manycore platform (32KB), for parallel applications like the image processing benchmark. The term 'dynamic' signifies the fact that architectures can be switched at runtime depending on the application characteristic as designed in 4.4. It is assumed that there are no switching penalties at least for this analysis. Next section delves on the effects of inclusion of switching penalties in time for any change in architecture. Microblaze processor is not considered for this analysis as it does not support executing large applications (that won't fit in its maximum supported memory of 128KB) from the DDR memory. The image processing benchmark and the hash benchmark are both ran on both the platforms and results are compiled to get insights. To run the hash benchmark on manycore platform, a core of the ρ-VEX dual-core platform with 32KB cache size is assumed as a core belonging to a ρ-VEX manycore platform as explained in the previous section.

Fig. 5.11 demonstrates the behavior of the DP for various DUs. The size of the image and hash table is varied along the x-axis from 1MP-10MP and 39.06KB-273.14KB respectively. Average execution times on all the platforms are shown on the y-axis. All the platform averages correspond to the average runtime of both benchmarks at the certain image size and hash table size. Fig. 5.11(a) shows the results for original platforms with no delay units added. It has a default delay of 15 cycles

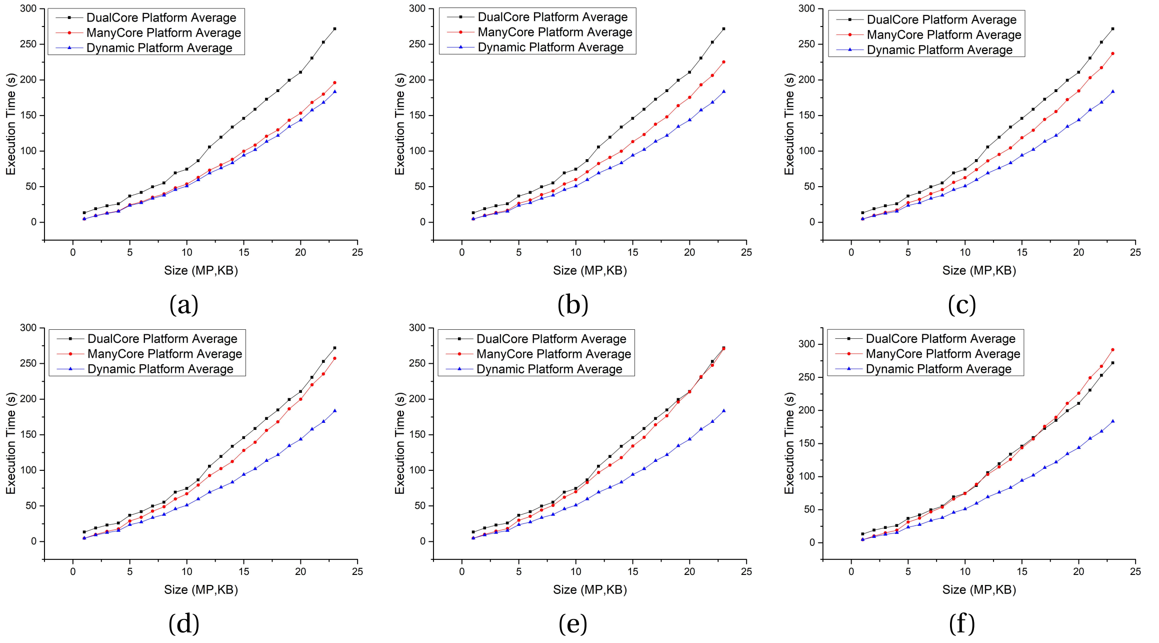


Figure 5.11: Behavior of the DP for (a)15 Delay Cycles (Original Configuration), (b)55 DUs, (c)75 DUs, (d)105 DUs, (e)125 DUs, and (f)155 DUs

from DDR memory to cache. It is clear from this figure that *the DP on an average performs better than the other two platforms.* The DP curve is very close to the manycore platform curve as the manycore platform gives a speed up to 7 in comparison to the dual-core platform (only single core is utilized) for image processing benchmark and the same speedup is not reciprocated in case of the hash benchmark. As the ρ-VEX bus handles only one memory request at a time, a single core of the dual-core platform is utilized here for the purposes of this experiment.

Figures 5.11(b) to 5.11(f) show the behavior of all the platforms for varying DUs. The curves of dual-core platform and DP do not vary much but the manycore platform curve overtakes the curve of the dual-core platform with an increase in DUs. The explanation for this behavior can be derived from the previous section (in particular, 5.2.3). Increase in the DUs increases the cache miss penalty for the manycore platform. With the increase in hash table size, cache misses increase and in turn, the cache miss penalty in time also increases. This is not a case for the dual-core platform as it can fit almost all the entries of the hash table because of its huge cache.

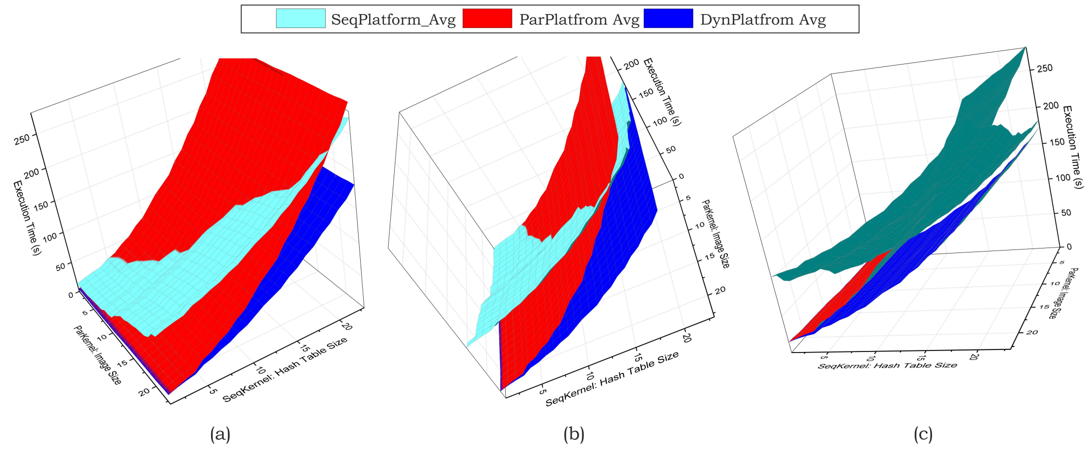


Figure 5.12: 3D views of the platform averages for all combinations of image size and hash table size, and for 155DUs.

Fig. 5.12 shows 3D views of the platform averages for 155DUs. Three different views are shown to get a complete picture. This is an extended version of Fig. 5.11(f) covering all combinations of image sizes and hash table sizes. It is clearly visible that the average performance curve of DP never touches the plane of either dual-core or many-core platforms although they appear quite close at the left-bottom area as the input sizes are very small in that region. These 3D plots prove once again that the DP performs better on average for sequential and parallel kernels than considering either sequential (dual-core) or parallel (manycore) platform alone. The speedup of DP ranges from 1.45 to 2.90 with respect to dual-core platform and 1.02 to 1.60 with respect to many-core platform depending on the input size (image size and hash table size). The average speedup of DP is 1.61 and 1.44 against dual-core and manycore platforms respectively for a considered input sizes.

5.4. IMPLICATIONS ON THE DYNAMIC PLATFORM

We have not considered any time penalty in initializing the platforms or switching architectures up till now. But it is important to understand how the time is distributed for various tasks. Table 5.2 shows the detailed information on varying time-consuming tasks. The timing is divided into two parts. First is the one time cost, which is a cost of reading the configuration bitstreams of the available architectures at the start of the DP. Second is the cost per architectural change, which is the cost incurred in time for downloading the bitstreams and binary files to PL and initializing the platforms. In total, it takes about 15ms to read the bitstreams stored in the micro-SD card into the program memory and about (200-280)ms for each architectural change. Initializing the ρ-VEX dual-core takes around 87ms in comparison to the ρ-VEX manycore platform which just takes less than a millisecond. This difference arises due to the fact that the dual-core platform needs the L2 cache to be flushed every time the code is flashed into the main memory and in addition, it is reseeded with the pseudo-random number generator (PRNG) as 'random' is a chosen cache eviction policy. Downloading the binary files take more time for manycore architecture than dual-core architecture. This difference is resulting from the fact that the dual-core system needs the code to be flashed to DDR memory once, whereas the manycore framework needs the code to be flashed directly to the instruction memory of all the cores.

The applications we considered earlier have runtime in terms of seconds. This means that the cost of time for architectural change, which is about 250ms here, can be neglected as long as the application runtime is much higher than this time. The significant contribution to the cost per architectural change is from downloading the configuration bitstream to FPGA. This downloading

Table 5.2: Timing costs involved in switching the architectures of the DP

One Time cost	Time for reading ρ-VEX manycore bitstream	15.367ms
	Time for reading ρ-VEX dual-core bitstream	15.455ms
Cost Per Architectural Change	Time to download the ρ-VEX manycore bitstream	196.567ms
	Time to download the ρ-VEX dual-core bitstream	193.365ms
	Time to initialize the ρ-VEX manycore Platform	0.317ms
	Time to initialize the ρ-VEX dual-core Platform	87.662ms
	Time to download the binary to ρ-VEX manycores	3.579ms
	Time to download the binary to ρ-VEX dual-core	0.046ms
	Total Time (ρ-VEX dual-core to ρ-VEX manycore)	*200.463ms*
	Total Time (ρ-VEX manycore to ρ-VEX dual-core)	*281.573ms*

time is bound to decrease in the future with further evolution in the FPGA technologies.

5.5. ρ-VEX Softcore vs ARM Hardcore

It is interesting to see the difference in performance between a softcore and the existing hardcore on the Zynq FPGA. The ρ-VEX dual-core is a softcore considered here and the on-chip ARM processor is the hardcore. The former one is operating at 50MHz frequency, and the latter one at a maximum of 667MHz frequency.

Fig. 5.13 shows the performances of the ρ-VEX softcore and the ARM hardcore. The equations to fit both the curves can be generated using polynomial interpolation. Using this interpolation, the ρ-VEX softcore follows the equation, Y=0.0045*$X^{2.002}$, whereas the ARM hardcore follows Y=0.0006*$X^{2.036}$. This approximately translates to ARM processor giving 7.5 speedup over the ρ-VEX processor. Though the frequency difference factor is ~13, the speedup achieved is nearly half of this value as the frequency of the Linux operating system varies widely and taking into account the 4-issue width of the ρ-VEX processor. This comparison shows that the advantage in having an optimized hard processor on-chip is the high-frequency operation they offer, which is difficult to replicate in softcores.

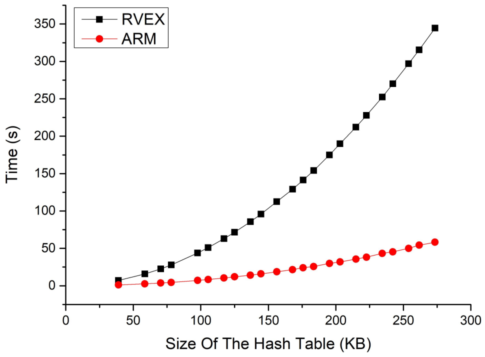


Figure 5.13: Performance comparision of ρ-VEX core (4-issue softcore) vs ARM core (Hardcore) for hash benchmark.

5.6. Discussion

After going through various parts of this thesis, the reader may have some important questions on several aspects of the overall design. This section tries to bridge this gap by discussing various topics (selected based on a common intuition) in an open and critical way.

5.6.1. Compatibility

Is the implemented Architecture compatible with respect to general FPGA? or just PYNQ?

Current implementation is dependent on the on-chip hard processor (ARM) to act as a central point in coordinating the execution of applications on PL. ARM processor usage can be eliminated if some dedicated logic is synthesized in the FPGA which can accept user inputs to determine which architecture needs to be used. Or, the code executed, currently, by the ARM processor can be modified to execute directly from the Xilinx environment/console which is connected to an FPGA.

5.6.2. Programming Languages

What are the programming languages that can be used with this architecture?

The ρ-VEX processor supports just 'C' language but the Microblaze processor supports both 'C' and 'C++' languages. Limited OpenCL language support for the ρ-VEX processor is provided in [53] but the results were not great.

5.6.3. Flexibility

How flexible is the proposed System?

The proposed system is flexible in a number of ways. The software framework allows to add any new softcore or modification to the existing softcores. The DDR memory can be allocated as per the application needs. The softcores are easily modifiable with a number of configuration options for various parameters such as size of the memory (cache and internal memory), number of cores and issue widths of each core.

5.6.4. Completeness

What makes the proposed system complete?

Features such as intelligent compilation to determine which architecture is needed, efficient memory management techniques and decreased runtime reconfiguration overhead are required in order to automate the whole process. Each topic in itself qualify to be an individual thesis subject. This is futher discussed in Section 6.3.

5.6.5. Softcores

Are there better softcores suited to this DP than the ones considered?

The DP, certainly, can be built using other softcores. However, the ones considered in this thesis are better suited as they are matured softcores and are easily available with a full support for an entire toolchain (TU Delft/Xilinx). There are some exciting new architectures such as Nu+ manycore [54] and scalable Pulp manycore [55] platforms that are yet to release to the open source community but have the desired features required to build the DP.

5.6.6. Coding Complexity

How easy is it to program a general application on this DP?

It certainly takes some time to get used to this platform as many things are done manually in the current framework such as dumping the binary directly to the cores of the processor and sending the address of the DDR memory to processors, each time it is allocated. However, the DP is easy to code once this barrier is overcome. Any new applications can be coded very easily for sequential architecture but for parallel architecture, consisting of manycores, care should be taken in dividing the code in such a way that they work independently of one another as there is no way to communicate between the cores of different streams in the current manycore architecture.

5.6.7. Cache Size Variation

Why is the L2 cache size varied and not the L1 cache size of the ρ-VEX dual-core platform?

The current implementation of L1 cache, which is split into blocks, one block per lanegroup, makes it very difficult to increase its size. Thus, the configurations used in this thesis have the L1 instruction and data cache size fixed at 4KB, and only L2 cache size is varied as it can be modified very easily to utilize the available BRAMs in the FPGA.

6

Conclusion and Future Work

If you don't know where you are going, any road will get you there.
- Lewis Carroll

This chapter is dedicated to summarizing the complete work done in this thesis and proposing future steps in a similar direction. First, a summary of all the chapters of the thesis is presented in Section 6.1. Subsequently, in Section 6.2, the main contributions are outlined to reflect on the goals discussed in the beginning of the thesis. Finally, the future work, which will be the improvement on this thesis, is proposed in Section 6.3.

6.1. Summary

Chapter 2 laid the foundation work for this thesis. First, the significance of RC was discussed. It was found that RC has a huge potential in saving energy consumption and providing power efficient performances. Then we looked at the challenges of RC in terms of tool support, reliability, runtime reconfiguration latency and memory architecture limitations. Further, the three processing architectures (CPU, GPU, and FPGA) along with their unique characteristics were discussed. Finally, the selection of softcores was elaborated. After discussing various trade-offs among the available softcores, the ρ-VEX manycore platform was selected for parallel architecture implementation, and the ρ-VEX dual-core and Microblaze platform were selected to implement sequential architecture.

In Chapter 3, the conceptual model was developed. The chapter started with the main motivation, which is derived from the extension of Amdahl's laws to the multicore era. In particular, this thesis is built on the third law, which, in summary, states that the given silicon area can be dynamically filled either with many small cores or one very big core, capable of occupying an area of all those small cores, in an effort to extract maximum performance. Further, the processing architectures (CPU, GPU) and the simple kernels (sequential: vector addition with induced dependency and parallel: vector dot product) developed for simulation were introduced.

Next, an attempt was made to model with real values. But the CPU performance was found to be dominating on average for both kernels over GPU. The model was biased towards CPU, as a result. The solution to build the model using reasonable performance values which on average perform similarly on CPU and GPU was then adopted. The modeling was successfully done and a number of observations were made. The important observation is that the DP performs better than CPU and GPU in the mid-region where the parallelism is neither too high nor low.

Chapter 4 detailed on the implementation of the DP. The ρ-VEX manycore platform, consisting of 10 cores, was implemented to cater to parallel needs of the applications. The Microblaze and the ρ-VEX dual-core platforms were implemented to fulfill the sequential requirements of the applications. All the platforms are implemented to work in cohesion with the on-chip ARM processor, which acts as a scheduler.

Chapter 5 discussed extensively on the experiments and results of the implemented softcores. First, the three benchmarks (Image processing, Hash and CRC) were introduced. These benchmarks are then used to evaluate all the three implemented architectures (ρ-VEX manycore, Microblaze and ρ-VEX dual-core). Evaluation of individual platforms against CPU was done and understood that the most of the performance difference occuring is mainly contributed to the high frequency of the CPU. This difference in performance is bound to be decreased in the future as the FPGA technology is continuously improving. The ρ-VEX dual-core platform was evaluated against variation in data paths from DDR memory to L2 cache for different cache sizes. The conclusion from this delay based parametric analysis is the FPGA area can be effectively converted into performance by increasing the size of the cache memory.

Then, the DP (consisting of ρ-VEX dual-core and ρ-VEX manycore) analysis was done with respect to image processing benchmark, acting as a parallel kernel and hash benchmark, as the sequential kernel. The results of this analysis comprehensively prove that the DP performs better on average than considering either sequential (ρ-VEX dual-core) or parallel platform (ρ-VEX manycore) alone. The average speedup of 1.61 and 1.44 was observed for the DP against sequential and parallel platforms respectively. The result was also plotted on a 3D surface to show that the DP performs well for varying levels of granularity in parallelism. Further, the implications of time penalty on the DP was examined. As the application runtime is significantly higher than the total time penalty of (200-280)ms per architectural change, the time penalty can be neglected in our case. Finally, the performance comparison of the ρ-VEX hardcore and the on-chip ARM processor was made. The result of this comparison shows that the ρ-VEX is about 7.5 times slower than the ARM core, and the performance difference is attributed to the higher frequency which is possible on hard processors as they are highly optimized ASICs.

6.2. Main Contributions

In Section 1.3, we looked at the research question and the goals of this thesis. There were four main goals that needed to be completed to address the research question. This section reflects on these goals and the main research question to see what was achieved, and how they were achieved. Further, a list of main contributions are mentioned.

The research question was:

> On average, for the sequential and parallel applications, can the performance on our designed architecture be better than on the CPU, the GPU or the static softcore?

The results, presented in Sections 5.2 and 5.3, addresses this question. Indeed, the results prove that the DP, our designed architecture, performs better on average for the sequential and parallel applications than the static softcore. The speedup of the dynamic platform ranges from 1.45 to 2.9 (average: 1.61) with respect to sequential platform and from 1.02 to 1.60 (average: 1.44) with respect to parallel platform. However, Sections 5.2.1 and 5.2.2 prove that in the current state of FPGA technology, the DP's performance is worse than both the CPU and GPU on average for both type of applications, the main reason being the lower frequency. From Section 3.4, it was understood that DP performs worse than CPU and GPU once the RF is increased beyond threshold. As the frequency difference factor of DP is more than 15 when compared to CPU and GPU, the Fig. 3.10 proves that the DP is worse in performance in the mid-region.

The research question was split into four goals to achieve the objective of this thesis. Let us understand how these goals were addressed in this thesis.

1. Model the performance for simple kernels by simulating them on CPU and GPU.

Two simple kernels, one (vector addition with induced dependency) favoring sequential architecture (CPU) and other (vector dot product) favoring parallel architecture (GPU), were simulated on both CPU and GPU. The average performances of kernels on each platform were projected on to a 3D plane to understand the platform behavior for various levels of parallelism exhibited by the underlying application. The DP was assumed to take the best of CPU and GPU performance values, and a realistic scenario was later injected into the modeling framework. Due to the limitation observed as a result of CPU bias on the kernels, the abstract model was then constructed using reasonable values which are derived from the earlier model.

2. Determine the region of interest in which the DP performs well and identify the feasibility of implementing such a dynamic architecture.

   From the previous point, it was clear that the abstract model was created to simplify the analysis and get meaningful insights on the behavior of the DP. Concept of Reduction Factor (RF) was introduced in Section 3.4. On increasing the RF from 1 to 10, it was observed that the DP performs better than CPU and GPU in the mid-region where the granularity in parallelism is neither too high nor too low. The feasibility of implementing such a dynamic architecture was carried out in Section 2.4, where the discussion took place to select the desired softcores for the DP. Three softcores (ρ-VEX dual-core, ρ-VEX manycore, and Microblaze) were selected to create the DP.

3. Designing and implementing a platform capable of switching between two architectures:

   - Parallel architecture to deal with the parallel applications.
   - Sequential architecture to deal with the sequential applications.

   Parallel architecture, namely, the ρ-VEX manycore consisting of 10 cores, which is scalable and capable of filling almost an entire Zynq PL was implemented. Sequential architecture, namely, the ρ-VEX dual-core and Microblaze platforms were implemented. The ρ-VEX dual-core platform was able to fill nearly an entire PL area whereas Microblaze was only able to occupy less than 1/5th of the PL space. The software for the DP, capable of switching between these two architectures at runtime, was also designed.

4. Performance analysis of the designed platform along with the discussion on the important trade-offs.

   Section 5.2 discussed the analysis of individual platforms with respect to three key benchmarks (Image processing, CRC, and hash). Section 5.3 elaborated on the DP analysis. The DP, capable of switching between sequential (ρ-VEX dual-core) and parallel (ρ-VEX manycore) architecture, was profiled for image processing (parallel) and hash (sequential) benchmarks. The results prove that it performs better on average than considering either of the static architecture alone. However, it does not perform better than CPU and GPU at the current state of the FPGA technology, mainly due to the frequency limitation. The timing implication on the switching architectures was considered, and concluded that it is negligible if the application runtime is significantly higher than 280ms, which is true for the benchmarks we considered.

The main contributions of this thesis are outlined as follows:

➤ Created a conceptual model of the dynamic processor to identify the region where it performs better than CPU and GPU, and to understand its limitations.

➤ Implemented three different architectures on Zynq FPGA to work in cohesion with the on-chip ARM processor, which acts as a scheduler here.

- ➤ Evaluated the performance of each architecture with respect to the sequential and parallel benchmarks.
- ➤ Implemented the DP which can switch between sequential and parallel architectures.
- ➤ Proved that the DP performs on an average better than the sequential or parallel platforms alone.
- ➤ Proved from the results that with the current state of FPGA technology, CPU and GPU perform better than the DP on average.

6.3. Future Work

There exists scope for improvements that can be done on the developed DP in this thesis. As this platform was built as a prototype to seek answers to our research question, surely it can be developed further. This section sheds some light on the possible enhancements that can add more value to the present work of this thesis.

6.3.1. ρ-VEX Improvements

As it is already pointed out in [39], the ρ-VEX architecture in the present stage needs at least three main improvements. First, the design of L1 cache needs to be revised as the current implementation makes it very difficult to increase its size. Second, the point mentioned in Section 2.1.4 about traditional Von-Neumann architectures becomes quite relevant here. The main limitation of this type of architecture is that the instruction streams are very memory-cycle-hungry, and the ρ-VEX, currently, can only issue one memory instruction per context per cycle which limits the available instruction level parallelism (ILP). Third and the final point relates to the ρ-VEX bus improvements. The ρ-VEX bus allows currently allows 4-byte transactions, and the instruction and data caches both utilize the same bus which means that the data reads from one context might have to wait for instruction reads from other context. This limits the bus performance as there is no scope for implementing modern bus features such as variable size transactions, bursts and out-of-order transactions which are commonly available in AXI bus implementation.

6.3.2. Single Processor Architecture

In this thesis, we considered two different versions of the ρ-VEX processor and the Microblaze processor, in addition. Currently, the limitation of manycore version is that it needs the code to be dumped to each core's instruction memory separately. To develop an ideal reconfigurable architecture, single-core and manycore softcore should use a single architecture. In this manner, the code can be easily developed uniformly for either a single-core or manycore architecture.

We considered mainly the VLIW architecture based processor (ρ-VEX) for DP analysis. Nowadays, an increasing number of RISC-V based cores are being developed by the research community. The DP can be built using these cores. The main challenge, however, is to vary the size of the RISC-V core for sequential and parallel needs. As explained in the background chapter, sequential applications need a big softcore with many modern CPU features such as OoO execution, branch prediction and big caches. On the other hand, parallel applications need simple but many cores that can exploit the available parallelism efficiently. These kind of softcores belonging to a single architecture do not exist yet and there is a big scope in developing these architectural frameworks in the future.

6.3.3. Automation

In the present version of this thesis, the selection of the processor, belonging to the DP, to execute an application is manual. However, this process can to be automated to make the DP choose its processor architecture automatically by parsing the user application.

The automation of the processor selection is easier said than done. First task is to write applications in a suitable programming language. OpenCL, or related programming frameworks are relevant as they contain useful information such as the number of threads the application uses. Then, the interpreter, using machine intelligence, needs to be written to analyze the user application to determine the suitable processor on which it can be run. This topic is easily worth an entire thesis.

6.3.4. Parameter based Architecture Selection

Parameter based selection of architecture adds another dimension to the previous enhancement, automation. Instead of selecting a processor just based on sequential or parallel needs of the application, a processor can be chosen based on additional details of the application. For instance, a sequential application can be based largely on control structures in which case branch prediction and OoO execution could come in handy. Or, a sequential application relies heavily on caches in which case huge caches can be useful. These requirements can be taken into account to create a pool of processor architectures with different capabilities. Once a certain behavior is detected in the user application, a suitable processor can be instantiated on the FPGA. Once again, this is a complicated task. However, knowing the applications beforehand can assist in developing a minimal logic based automation.

BIBLIOGRAPHY

[1] M. D. Hill and M. R. Marty, *Amdahl's law in the multicore era,* Computer **41** (2008).

[2] J. Cardoso and M. Hübner, *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign* (Springer Science & Business Media, 2011).

[3] M. Pickavet, W. Vereecken, S. Demeyer, P. Audenaert, B. Vermeulen, C. Develder, D. Colle, B. Dhoedt, and P. Demeester, *Worldwide energy needs for ict: The rise of power-aware networking,* in *Advanced Networks and Telecommunication Systems, 2008. ANTS'08. 2nd International Symposium on* (IEEE, 2008) pp. 1–3.

[4] G. Fettweis and E. Zimmermann, *Ict energy consumption-trends and challenges,* in *Proceedings of the 11th international symposium on wireless personal multimedia communications,* Vol. 2 ((Lapland, 2008) p. 6.

[5] *Why are computers important in the world?* http://www.answers.com/Q/Why_are_computers_important_in_the_world, (Accessed on 2018-06-30).

[6] Y. C. Lee and A. Y. Zomaya, *Interweaving heterogeneous metaheuristics using harmony search,* in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on* (IEEE, 2009) pp. 1–8.

[7] J. Rabaey, *Low power design essentials* (Springer Science & Business Media, 2009).

[8] T. A. Claasen, *High speed: not the only way to exploit the intrinsic computational power of silicon,* in *Solid-State Circuits Conference, 1999. Digest of Technical Papers. ISSCC. 1999 IEEE International* (IEEE, 1999) pp. 22–25.

[9] F. Birol, *Leave oil before it leaves us,* http://www.youtube.com/watch?v=m377Is4tGF0, (Accessed on 2018-06-30).

[10] H. Simon, *Leibniz-rechenzentrum,* TU Munich (2009).

[11] R. Hartenstein, *Why we need reconfigurable computing education; 1st int'l workshop on reconfigurable computing education (rc education 2006), march 1, 2006,* KIT Karlsruhe Insitute of Technology, Germany .

[12] K. Gaj and T. El-Ghazawi, *Cryptographic applications,* RSSI Reconfigurable Systems Summer Institute (2005).

[13] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach* (Elsevier, 2011).

[14] E. Peláez, *Parallelism and the crisis of von neumann computing,* Technology in Society **12**, 65 (1990).

[15] M. Flynn and O. Mencer, *Accelerating computations using data parallelism,* in *CE TU Delft colloquium* (2009).

[16] I. Sourdis and G. N. Gaydadjiev, *Hipeac: Upcoming challenges in reconfigurable computing,* in *Reconfigurable Computing* (Springer, 2011) pp. 35–52.

[17] G. N. Gaydadjiev and S. Vassiliadis, *Flux caches: What are they and are they useful?* in *International Workshop on Embedded Computer Systems* (Springer, 2005) pp. 93–102.

[18] V. Degalahal and T. Tuan, *Methodology for high level estimation of fpga power consumption,* in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference* (ACM, 2005) pp. 657–660.

[19] *Tick–tock model,* https://en.wikipedia.org/wiki/Tick\T1\textendashtock_model, (Accessed on 2018-06-30).

[20] *How much information?* http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/, (Accessed on 2018-06-30).

[21] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, *et al., Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu,* ACM SIGARCH computer architecture news **38**, 451 (2010).

[22] J. O. Hamblen, T. S. Hall, and M. D. Furman, *Rapid prototyping of digital systems: SOPC edition*, Vol. 1 (Springer Science & Business Media, 2007).

[23] *Introduction to fpga design for embedded systems,* https://www.coursera.org/learn/intro-fpga-design-embedded-systems, (Accessed on 2018-06-30).

[24] U. Farooq, Z. Marrakchi, and H. Mehrez, *Fpga architectures: An overview,* in *Tree-based Heterogeneous FPGA Architectures* (Springer, 2012) pp. 7–48.

[25] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. A. El-Ayat, and A. Mohsen, *An architecture for electrically configurable gate arrays,* IEEE Journal of Solid-State Circuits **24**, 394 (1989).

[26] F. R. Carter W, Duong K and S. S, *A user programmable reconfiguration gate array,* in *IEEE Custom Integrated Circuits Conference* (IEEE, May 1986) pp. 233—-235.

[27] S. C. Wong, H. So, J. H. Ou, and J. Costello, *A 5000-gate cmos epld with multiple logic and interconnect arrays,* in *Custom Integrated Circuits Conference, 1989., Proceedings of the IEEE 1989* (IEEE, 1989) pp. 5–8.

[28] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*, Vol. 497 (Springer Science & Business Media, 2012).

[29] A. DeHon, *Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% lut utilization),* in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays* (ACM, 1999) pp. 69–78.

[30] *Stratix: Stratix ii fpgas,* https://www.altera.com/products/fpga/stratix-series/stratix-ii/overview.html (), (Accessed on 2018-06-30).

[31] *Altera: Flex 10k embedded programmable logic device family,* http://www.altera.com/literature/ds/dsf10k.pdf (), (Accessed on 2018-06-30).

[32] M. A. E. R. W. S. Louise H. Crockett, Ross A. Elliot, *The Zynq Book* (Strathclyde Academic Media, 2014).

[33] *Mipsfpga getting started guide 2.0*, https://mips.com/downloads/mipsfpga-getting-started-guide-2-0/, (Accessed on 2018-06-30).

[34] *Pulpino : Open source 1-core microcontroller system*, https://github.com/pulp-platform/pulpino, (Accessed on 2018-06-30).

[35] *Microblaze processor reference guide*, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug984-vivado-microblaze-ref.pdf, (Accessed on 2018-06-30).

[36] S. Wong and F. Anjam, *The delft reconfigurable vliw processor*, system **1**, 3 (2009).

[37] *The hp vex toolchain*, http://www.hpl.hp.com/downloads/vex/, (Accessed on 2018-06-30).

[38] J. Hoozemans, *Targeting static and dynamic workloads with a reconfigurable VLIW processor*, Ph.D. thesis, Delft University of Technology (2018).

[39] J. Dirks, *Efficient inter-thread communication on the reconfigurable ρ-VEX*, Master's thesis, Delft University of Technology (2018).

[40] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, *et al.*, *Openpiton: An open source manycore research framework*, in *ACM SIGARCH Computer Architecture News*, Vol. 44 (ACM, 2016) pp. 217–232.

[41] D. Anderson and T. Shanley, *Pentium processor system architecture* (Addison-Wesley Professional, 1995).

[42] S. Wallentowitz, P. Wagner, M. Tempelmeier, T. Wild, and A. Herkersdorf, *Open tiled manycore system-on-chip*, arXiv preprint arXiv:1304.5081 (2013).

[43] *Lisnoc: A free network-on-chip implementation*, http://www.lisnoc.org/, (Accessed on 2018-06-30).

[44] *The multicore association (mca)*, https://www.multicore-association.org, (Accessed on 2018-06-30).

[45] R. Busseuil, L. Barthe, G. M. Almeida, L. Ost, F. Bruguier, G. Sassatelli, P. Benoit, M. Robert, and L. Torres, *Open-scale: A scalable, open-source noc-based mpsoc for design space exploration*, in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on* (IEEE, 2011) pp. 357–362.

[46] G. Marchesan Almeida, G. Sassatelli, P. Benoit, N. Saint-Jean, S. Varyani, L. Torres, and M. Robert, *An adaptive message passing mpsoc framework*, International Journal of Reconfigurable Computing **2009** (2009).

[47] *Axi hwicap v3.0*, https://www.xilinx.com/support/documentation/ip_documentation/axi_hwicap/v3_0/pg134-axi-hwicap.pdf, (Accessed on 2018-06-30).

[48] *Thesis-materials: Prashanth g l*, https://bitbucket.org/guledal-prashanth/thesis-materials_prashanthgl/src/master/, (Accessed on 2018-08-09).

[49] *Hash functions*, http://www.cse.yorku.ca/~oz/hash.html, (Accessed on 2018-06-30).

[50] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, *Mibench: A free, commercially representative embedded benchmark suite*, in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on* (IEEE, 2001) pp. 3–14.

[51] *Wcet benchmarks,* http://www.mrtc.mdh.se/projects/wcet/benchmarks.html, (Accessed on 2018-06-30).

[52] S. Hilmarsson, *Hala ρ-VEX: Highly-Programmable Dynamically-Reconfigurable FPGA-based Streaming Platform for Image Processing,* Master's thesis, Delft University of Technology (2018).

[53] *Almarvi 621439,* http://www.almarvi.eu, (Accessed on 2018-06-30).

[54] *Nu+: A deeply configurable, open-source gpu-like processor core,* http://nuplus.hol.es/index.html, accessed: 2018-06-30.

[55] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, *Pulp: A parallel ultra low power platform for next generation iot applications,* in *Hot Chips 27 Symposium (HCS), 2015 IEEE* (IEEE, 2015) pp. 1–39.

A

Script based Microblaze Processor Design

TCL Script to generate the Microblaze Processor working in cohesion with the Zynq Processing System.

```tcl

## author: Prashanth G L
## Hardware : Microblaze Processor design with Zynq Processing system


#############################################################################
################### Design using 8,16,32,64KB memory ###################
#############################################################################

# Create a project in Vivado 2016.2 (for other versions, slight modifications are required as the IPs are updated/changed) for your required board configuration and then open the tcl command prompt and execute the below commands.

# 1. First create a Zynq processing system:

create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5 processing_system7_0
apply_bd_automation -rule xilinx.com:bd_rule:processing_system7 -config {make_external "FIXED_IO, DDR" Master "Disable" Slave "Disable" }  [get_bd_cells processing_system7_0]
set_property -dict [list CONFIG.PCW_FPGA0_PERIPHERAL_FREQMHZ {100} CONFIG.PCW_USE_S_AXI_HP0 {1}] [get_bd_cells processing_system7_0]

# 2. Now create a Microblaze processor

create_bd_cell -type ip -vlnv xilinx.com:ip:microblaze:9.6 microblaze_0
set_property -dict [list CONFIG.G_TEMPLATE_LIST {2} CONFIG.C_USE_ICACHE {0} CONFIG.C_USE_DCACHE {0} CONFIG.C_D_AXI {1} CONFIG.C_USE_MSR_INSTR {1} CONFIG.C_USE_PCMP_INSTR {1} CONFIG.C_USE_BARREL {1} CONFIG.C_USE_DIV {1} CONFIG.C_USE_HW_MUL {2} CONFIG.C_USE_FPU {2} CONFIG.C_CACHE_BYTE_SIZE {32768} CONFIG.C_ICACHE_LINE_LEN {8} CONFIG.C_ICACHE_VICTIMS {8} CONFIG.C_ICACHE_STREAMS {1} CONFIG.C_DCACHE_BYTE_SIZE {32768} CONFIG.C_DCACHE_LINE_LEN {8} CONFIG.C_DCACHE_USE_WRITEBACK {1} CONFIG.C_DCACHE_VICTIMS {8} CONFIG.C_MMU_ZONES {2} CONFIG.C_USE_BRANCH_TARGET_CACHE {1}] [get_bd_cells microblaze_0]

# 3. Uncomment one of the below options to select the required memory (unified instruction and data memory). 128KB memory version is uncommented here as an example.

# apply_bd_automation -rule xilinx.com:bd_rule:microblaze -config {local_mem "8KB" ecc "None" cache "None" debug_module "Debug Only" axi_periph "Enabled" axi_intc "0" clk "/processing_system7_0/FCLK_CLK0 (100 MHz)" }  [get_bd_cells microblaze_0]
# apply_bd_automation -rule xilinx.com:bd_rule:microblaze -config {local_mem "16KB" ecc "None" cache "None" debug_module "Debug Only" axi_periph "Enabled" axi_intc "0" clk "/processing_system7_0/FCLK_CLK0 (100 MHz)" }  [get_bd_cells microblaze_0]
# apply_bd_automation -rule xilinx.com:bd_rule:microblaze -config {local_mem "32KB" ecc "None" cache "None" debug_module "Debug Only" axi_periph "Enabled" axi_intc "0" clk "/processing_system7_0/FCLK_CLK0 (100 MHz)" }  [get_bd_cells microblaze_0]
# apply_bd_automation -rule xilinx.com:bd_rule:microblaze -config {local_mem "64KB" ecc "None" cache "None" debug_module "Debug Only" axi_periph "Enabled" axi_intc "0" clk "/processing_system7_0/FCLK_CLK0 (100 MHz)" }  [get_bd_cells microblaze_0]
apply_bd_automation -rule xilinx.com:bd_rule:microblaze -config {local_mem "128KB" ecc "None" cache "None" debug_module "Debug Only" axi_periph "Enabled" axi_intc "0" clk "/processing_system7_0/FCLK_CLK0 (100 MHz)" }  [get_bd_cells microblaze_0]


# 4. Now Modify the BRAM ports to give one connection to ZYNQ and the other to the Microblaze Processor

delete_bd_objs [get_bd_intf_nets microblaze_0_local_memory/microblaze_0_dlmb_bus] [get_bd_intf_nets microblaze_0_local_memory/microblaze_0_dlmb_cntlr] [get_bd_cells microblaze_0_local_memory/dlmb_bram_if_cntlr]
set_property -dict [list CONFIG.C_NUM_LMB {2}] [get_bd_cells microblaze_0_local_memory/ilmb_bram_if_cntlr]
connect_bd_intf_net [get_bd_intf_pins microblaze_0_local_memory/dlmb_v10/LMB_Sl_0] [get_bd_intf_pins microblaze_0_local_memory/ilmb_bram_if_cntlr/SLMB1]
create_bd_intf_port -mode Slave -vlnv xilinx.com:interface:bram_rtl:1.0 BRAM_PORTA
set_property CONFIG.MASTER_TYPE [get_property CONFIG.MASTER_TYPE [get_bd_intf_pins microblaze_0_local_memory/lmb_bram/BRAM_PORTA]] [get_bd_intf_ports BRAM_PORTA]
connect_bd_intf_net [get_bd_intf_pins microblaze_0_local_memory/lmb_bram/BRAM_PORTA] [get_bd_intf_ports BRAM_PORTA]
delete_bd_objs [get_bd_intf_nets BRAM_PORTA_1]
delete_bd_objs [get_bd_intf_ports BRAM_PORTA]

# 5. Now Connect Zynq master and slave axi clocks

connect_bd_net [get_bd_pins processing_system7_0/S_AXI_HP0_ACLK] [get_bd_pins processing_system7_0/FCLK_CLK0]
connect_bd_net [get_bd_pins processing_system7_0/M_AXI_GP0_ACLK] [get_bd_pins processing_system7_0/FCLK_CLK0]

# 6. Instantiate and configure the AXI Bram controller

create_bd_cell -type ip -vlnv xilinx.com:ip:axi_bram_ctrl:4.0 axi_bram_ctrl_0
set_property -dict [list CONFIG.SINGLE_PORT_BRAM {1}] [get_bd_cells axi_bram_ctrl_0]
connect_bd_intf_net [get_bd_intf_pins axi_bram_ctrl_0/BRAM_PORTA] -boundary_type upper [get_bd_intf_pins microblaze_0_local_memory/BRAM_PORTA]

# 7. Apply the automation to connect the AXI Bram controller instantiated in the previous step to Zynq Processing System

apply_bd_automation -rule xilinx.com:bd_rule:axi4 -config {Master "/microblaze_0 (Periph)" Clk "Auto" }  [get_bd_intf_pins processing_system7_0/S_AXI_HP0]
apply_bd_automation -rule xilinx.com:bd_rule:axi4 -config {Master "/processing_system7_0/M_AXI_GP0" Clk "Auto" }  [get_bd_intf_pins axi_bram_ctrl_0/S_AXI]

# 8. Optimize the routing and regenerate layout

regenerate_bd_layout -routing
regenerate_bd_layout

# 9. Assign correct addresses to avoid any conflicts with the address range
assign_bd_address [get_bd_addr_segs {microblaze_0_local_memory/ilmb_bram_if_cntlr/SLMB1/Mem }]
set_property offset 0x20000000 [get_bd_addr_segs {microblaze_0/Data/SEG_processing_system7_0_HP0_DDR_LOWOCM}]

```

```
# 10. Validate the design

validate_bd_design

# 11. Final step is to generate the Bit file (modify the path to the sources to suit to your environment setup).

make_wrapper -files [get_files /home/<user_name>/mblaze_design/mblaze_design.srcs/sources_1/bd/design_1/design_1.bd] -top
save_bd_design
reset_run synth_1
launch_runs impl_1 -to_step write_bitstream
```