EFFICIENT HARDWARE ACCELERATION ON SOC-FPGA WITH OPENCL

by

Susmitha Gogineni

APPROVED BY SUPERVISORY COMMITTEE:

Benjamin Carrion Schafer, Chair

Dinesh K. Bhatia

Mehrdad Nourani

Dedicated to my Parents, Grandparents and Teachers.

EFFICIENT HARDWARE ACCELERATION ON SOC-FPGA WITH OPENCL

by

SUSMITHA GOGINENI, B.Tech.

THESIS

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN

ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS

August 2017

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my thesis adviser Dr. Benjamin Carrion Schafer for his guidance and support in my research. I have benefited tremendously from his deep theoretical knowledge, rich practical experience and research attitude. He always was there to help motivate me, and I appreciate his enthusiasm and patience in helping students. Under his guidance, I had a remarkable self improvement. Thank you, Professor!

I would thank Dr. Dinesh Bhatia and Dr. Mehrdad Nourani for being on my committee and reviewing my work. I would like to thank the Department of Electrical Engineering and Computer Engineering at The University of Texas at Dallas and Design Automation & Reconfigurable Computing Lab (DARC Lab) for providing resources and creating an incredible research environment for students like me. I heartily thank my lab mates Farah, Mihir, Siyuan, Pandy and Nandeesh for their valuable suggestions, help and moral support. I am thankful to Rohit Somwanshi from IDEA Lab, for the suggestions and inspiration he gave me through out my research process. I need to specially thank Sussannah Martin from Intel, for providing me training and support in using the devices and resources for Experimentation. I also appreciate the Office of Graduate Studies (OGS) for their help in my research work.

My heartfelt thanks to my Family and my Extended Family in Dallas, who believed in my interests and encouraged me. They have done the best they can do for my success. I would thank my roommates and friends who are my family at UTD for their care and encouragement.

August 2017

EFFICIENT HARDWARE ACCELERATION ON SOC-FPGA WITH OPENCL

Susmitha Gogineni, MSEE
The University of Texas at Dallas, 2017

Supervising Professor: Benjamin Carrion Schafer, Chair

Field Programmable Gate Arrays (FPGAs) are taking over the conventional processors in the field of High Performance computing. With the advent of FPGA architectures and High level synthesis tools, FPGAs can now be easily used to accelerate computationally intensive applications like, e.g., AI and Cognitive computing. One of the advantages of raising the level of hardware design abstraction is that multiple configurations with unique properties (*i.e.* area, performance and power) can be automatically generated without the need to re-write the input description. This is not possible when using traditional low-level hardware description languages like VHDL or Verilog. This thesis deals with this important topic and accelerates multiple computationally intensive applications amiable to hardware acceleration and proposes a fast heuristic Design Space Exploration method to find dominant design solutions quickly.

In particular, in this work, we developed different computationally intensive applications in OpenCL and mapped them onto a heterogeneous SoC-FPGA. A Genetic Algorithm (GA) based meta-heuristics that does automatic Design Space Exploration (DSE) on these applications was also developed as GA has shown in the past to lead to good results in multi-objective optimization problems like this one. The developed explorer automatically inserts a set of control knobs into the OpenCL behavioral description, e.g., to control how to synthesize loops (unroll or not), and to replicate Compute Units (CUs). By tuning the these control

attributes with possible values, thousands of different micro-architecture configurations can be obtained. Thus, an exhaustive search is not feasible and other heuristics are needed. Each configuration is compiled using Altera OpenCL SDK tool and executed on Terasic DE1-SoC FPGA board platform to record the corresponding performance and logic utilization. In order to measure the quality of the proposed GA-based heuristic, each application is explored exhaustively (taking multiple days to finish for smaller designs) to find the dominant optimal solutions (Pareto Optimal Designs). For complex and larger designs, exploring the entire design space exhaustively is not feasible due to very large design space. The comparison is quantified by using metrics like Dominance, Average Distance from Reference Set (ADRS) and run time speed up, showing that our proposed heuristics lead to very good results at a fraction of the time of the exhaustive search.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

There has always been a quest for performance and power efficient hardware to meet the growing processing needs. This developed trends in hardware architectures as per Moores Law [7] with the increase in the number of transistor density and frequency. Nevertheless with the breakdown of Dennard's scaling in the mid 2000's it is no more feasible to increase frequency due to power constraints [8]. This led to a paradigm shift in computer architecture where alternatives like Multi-Cores, Graphic Processing Units (GPUs) and heterogeneous SoC with dedicated hardware accelerators are being developed. Todays applications involve computing exponential amount of data in less time and also real time systems require ultra low-latency responses. Examples include, Artificial Intelligence (AI), Cognitive Computing and Cloud Computing. Therefore, there is an indispensable quest for hardware acceleration and power efficient systems.

1.1 FPGA-based Hardware Acceleration

FPGAs support computational acceleration as they can provide performance and power efficiency. Moreover, FPGAs can be integrated with other systems like CPUs, GPUs. Recent studies prove that, platforms with combination of FPGA and CPU achieve better performance than CPUs alone on certain workloads. Researches have shown that by offloading a CPU from computationally intensive parts to an FPGA, significant acceleration can be achieved [[9],[10]]. This idea is picked up and implemented by leading FPGA vendors as well.

In 2015, Intel acquired Altera to encourage FPGA products in the data center and IoT markets [11]. Microsoft started project Catapult in 2010, a hyper scale acceleration fabric technology which integrates an FPGA into Microsoft data centers to accelerate networking,

security, cloud services and artificial intelligence [12]. The project Amazon EC2 F1, uses customizable FPGA designs to accelerate computing in Amazon Web Services (AWS). In 2017, Baidu has deployed Xilinx FPGA-based application acceleration services in their public cloud [13].

1.2 Strengths of FPGA-based Hardware Acceleration

In this section we discuss the benefits of FPGA for hardware acceleration and efficient data processing.

1. Re-Configurable:

   FPGA is a programmable hardware and can be quickly reprogrammed within milliseconds. They can be programmed to any dedicated hardware function. This gives them the flexibility to be continually updated in the field.

2. Moore's Law:

   As per the prediction of Gordon Moore in 1965, the number of transistors on the Integrated chip have been doubling approximately every two years [7]. This also holds true for FPGAs enabling complete hardware systems to be implemented on them with minimal off-chip resources. Moore's law has enabled FPGAs to be integrated with other hardware units like CPUs, GPUs, DSPs, ASICs and Memory on the same chip. This supports division of workload and parallel processing on FPGAs.

3. High level Synthesis(HLS) to program FPGA:

   Programming methods of FPGA have come a long way. Initially, FPGA programming was relegated to only hardware designers skilled in low level languages like Verilog, VHDL, which was also a time consuming process. With the increase in complexity of FPGAs, most of the FPGA vendors now provide HLS tools to synthesize High level languages (e.g., C, C++, SystemC, OpenCL) to programmable logic. This makes

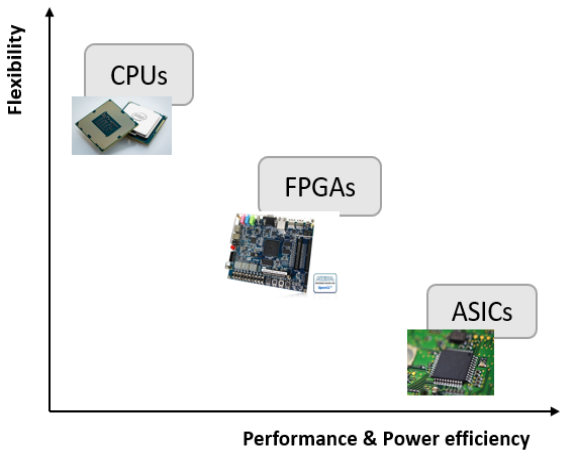


Figure 1.1. Flexibility and Power Efficiency trade-off for CPUs, FPGAs and ASICs

programming an FPGA easier and faster. Moreover, HLS allows Design Space Exploration as it allows to generate different architectures without the need to modify the behavioral description.

4. Cost Effective and Rapid Prototyping:

   FPGAs provide significant results in reducing time to market and development costs. FPGAs are claimed as the cost effective alternatives to ASICs as NRE costs are negligible.

5. Power Effective:

   FPGAs provide lower power foot prints than Processors, due to dedicated hardware Implementations (see Figure 1.1). Data can be processed in parallel at a much lower frequency faster than with CPUs.

1.3 Challenges of FPGA for Hardware Acceleration

This section discusses the limitations of FPGA.

1. Communication Overhead

   In heterogeneous systems like SoC-FPGA, the parallel processing efficiency is limited due to data transfer at the interface. Therefore, we need architectures to handle communication overhead, which is one of the major factors impacting the speed-up.

2. Programming the FPGA

   Though, we have reached a point where we can synthesize most of the popular High level language descriptions onto FPGAs. there are still some limitations. The programmer needs to be knowledgeable enough about the hardware Micro-Architectures and the compiler tools. The compilers have a complex compilation process with a number of stages and cycles, that can take several hours to synthesize a single design.

1.4 Motivation

Growing performance needs for applications demand faster and efficient hardware. Since 2006, with the breakdown of Dennard's scaling [8], a paradigm shift in computer architecture is taking place. The term dark silicon has been coined to describe the part of the circuitry of an Integrated Circuit (IC) that cannot be powered for a given thermal design constraint [14]. It is estimated that at 22 nm, 32% of a fixed-size IC must be powered off and at 8nm more than 50% . One solution that is being proposed is to customize the computing platforms to the application domain, also known as domain specific computing. At the computer level, by using CPUs with FPGA [15, 16], at the chip level through heterogeneous Multiprocessor Systems on-Chips (MPSoCs) with dedicated Hardware Accelerators (HWacc) [17], and at the micro-architectural level by being able to quickly re-optimize micro-architectures for different computing platforms. The acquisition of Altera by Intel is also mainly motivated by this, and Intel has recently announced that it is targeting the production of around 30% of the servers with FPGAs in data-centers by 2020 [16].

The main problem when adapting these computing systems to different domains is that, it is driving IC design to complexities never seen before, making their design extremely challenging. At the same time, the increasing demand for newer products with more powerful features, new or revised versions of these systems (i:e: SoCs) need to be taped-out in shorter and shorter time frames. This means that time-to-market, but also time-in-the-market is getting shorter and hence the risk of not achieving the estimated return on investment (ROI) is growing significantly. To address these issues, companies are relying on third Party Intellectual Properties (3PIPs) to meet their schedules. They have also started to rely on High-Level Synthesis (HLS) to increase their design productivity. The International Technology Roadmap for Semiconductors (ITRS) already suggested in 2013 that by 2020 a 10x productivity increase for designing complex SoCs is needed. Two main factors were predicted to help achieving this goal. The first is the re-use of components. ITRS estimates that around 90% of the SoCs will be composed of re-used components. Secondly, the use of new design methodologies to raise the level of abstraction i:e: HLS, also called C-based VLSI design [18].

Raising the level of design abstraction has multiple advantages. One particular advantage that this thesis aims at exploiting is the ability to generate different micro-architectures without having to modify the behavioral description, also known as design space exploration (DSE). This is typically done by changing different synthesis options to e.g., control loop unroll factor, Number of duplicated compute units etc. HLS DSE can be classified as a multi-objective optimization problem. The presence of multiple objectives in HLS DSE, gives rise to a set of optimal solutions, also known as Pareto-optimal solutions, instead of a single optimal solution. In the absence of any further information, none of these Pareto-optimal solutions can be considered to be better than the other. This distinct advantage of C-based VLSI design is also a weakness as it forces users to fully understand each exploration knob, which is often tool dependent. In addition, the search space is extremely large. For a simple

8-TAP FIR filter the solution space consists of over 1000 possible knob settings. Each time a new configuration is generated, this has to be synthesized in order to extract the different design metrics (i.e., area, latency, delay). Thus, fast, automated methods to evaluate the effects of these knobs are required.

1.5 Thesis Structure

Chapter 2, presents a brief study on the evolution of FPGAs and SoC-FPGA. Chapter 3, introduces High Level Synthesis (HLS) and OpenCL parallel programming Language. Chapter 4, introduces the methodology and architectures used for hardware acceleration on FPGAs. Then we describe the applications described in OpenCL and the acceleration results obtained. In Chapter 5, we describe the methodology to perform automatic DSE using and exhaustive search and a fast heuristic approach based on GA. This chapter measure the Quality of Results (QoR) of our proposed heuristic using standard metrics. This section also discusses the results and makes a few observations from the results. Finally, chapter 6, presents the conclusions and discusses some future work.

CHAPTER 2

FIELD PROGRAMMABLE GATE ARRAYS (FPGAS)

2.1 Overview of Programmable Logic

Early programmable devices had architectural regularity and functional flexibility but by the mid-1960s, field-programmability, the ability to change the logic function of a chip after the fabrication process, was achieved via the introduction of cutpoint cellular arrays [19]. The functionality of each logic cell in array would be determined by setting the programmable fuses through currents or photo-conductive exposure, although all the connections within an array were fixed. Hence, field programmability was introduced which allowed simplified array manufacturing and wider applicability.

In 1970s, read only memory (ROM)-based programmable devices were introduced, although mask programmable and fuse-programmable ROMs (PROMs) with n-input addresses can be used to implement n-input logic functions, there were issues with area-efficiency that were introduced with these devices. Hence, later developments were made to introduce programmable logic arrays (PLAs) where each plane in a wired-AND or wired-OR structure along with inverters can build any AND or OR logic term. This type of structure closely matches common logic functions and is more area efficient. The architectures evolved further in 1977 by Monolithic Memories Incorporated (MMI) with the realization that sufficient flexibility was provided by programmable AND-plane followed by fixed OR-plane, in devices which are called programmable array logic (PAL) [20]. Figure 2.1 shows comparison of PLA and PAL architectures. It can be seen that in Figure 2.1b, the combinational logic is fed to fixed sequential logic in the form of D-type flip flop macro cells. PLA, PAL and generic array logic (GAL) can be classified as Low-density PLDs.

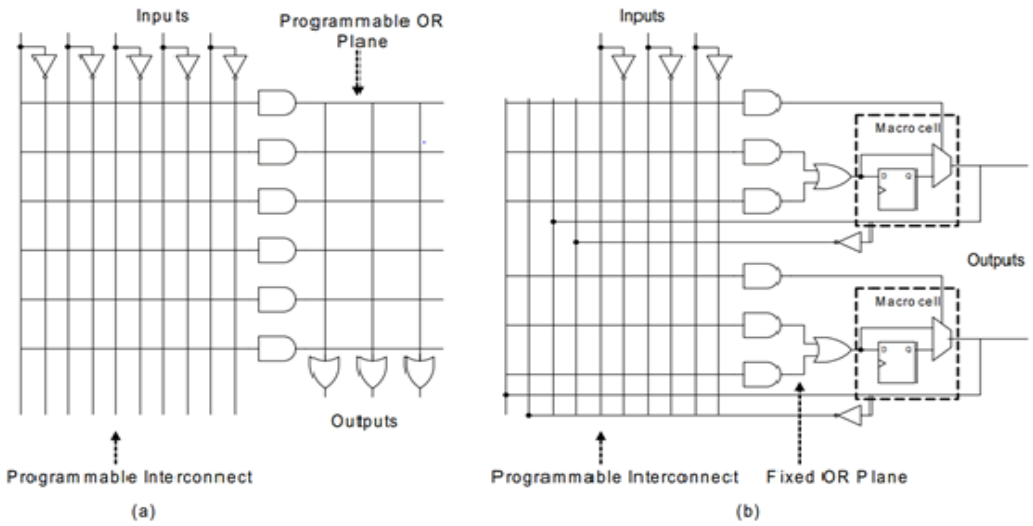


Figure 2.1. Architecture of PLA and PAL

2.2 SRAM-based FPGA

FPGAs can be classified as High-density PLDs. The first static memory-based FPGA (commonly called an SRAM based FPGA) architecture allowed for both logic and interconnection configuration using a stream of configuration bits. Unlike its contemporary cellular array counterparts, both wide-input logic functions and storage elements could be implemented in each logic cell. Additionally, the programmable inter-cell connections could be easily changed through memory-configurability to enable the implementation of a variety of circuit topologies. Although static memory offers the most flexible approach to device programmability, it requires a significant increase in area per programmable switch compared to ROM implementations [20]. SRAM-based FPGAs can be programmed using master and slave mode. In master mode, the FPGA reads an external flash memory chip to configure the bitstream. However, in slave mode, the FPGA is configured by an external master device such as a processor. SRAM based chips include most chips from Xilinx - Virtex Spartan families and Altera - Stratix Cyclone [21].

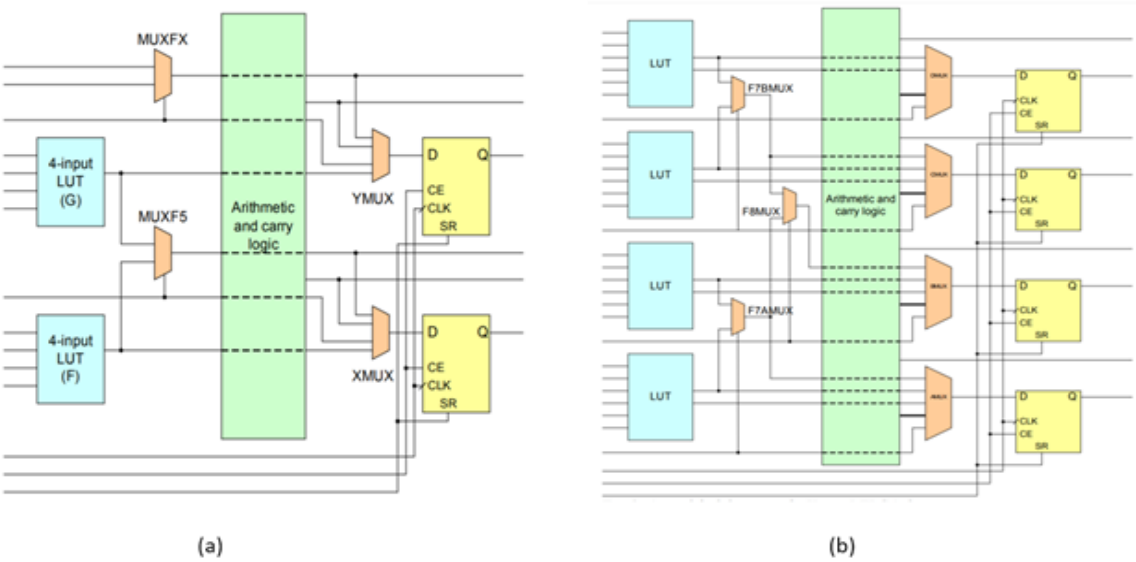


Figure 2.2. Xilinx Slices (a) Virtex-4 and earlier (b)Virtex-5 [1].

2.3 Modern FPGA Logic Cells: Xilinx Slices vs Altera ALM

FPGAs consists of multiple programmable logic blocks called Logic Cells. Logic cells in Xilinx is called a slice and in Altera it is called Adaptive Logic Module(ALM). The architecture of Xilinx Slices used in Virtex-4 and earlier, as shown in Figure 2.2(a), include two 4-input LUTs, two dedicated user-controlled multipliers for combinational logic, dedicated arithmetic logic gates for fast and efficient multiplication and two 1-bit registers that can be configured either as flip-flops or as latches. On the other side, as shown in Figure 2.2(b), Virtex-5 Slices include four LUTs that can be configured as 6-input LUTs with 1-bit output or 5-input with 2-bit output, three dedicated user controlled multipliers for combinational logic, dedicated arithmetic logic and four 1-bit registers that can be configured either as flip-flops or as latches. The multiplexers are not actually user controlled however the path is selected during FPGA Programming [1].

Altera FPGAs of Stratix and Cyclone families, as shown in Figure 2.3, uses slightly different logic blocks called Adaptive Logic Modules (ALM), which include two 4-input LUTs and four 3-input LUTs for combinational logic implementation, dedicated arithmetic

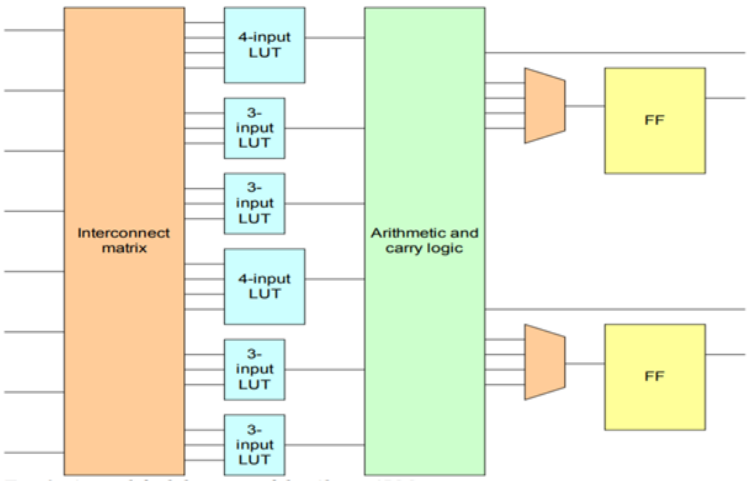


Figure 2.3. Altera ALM in Stratix and Cyclone families [1].

and carry logic and two programmable registers. ALM has 8 inputs out of which some inputs are connected to more than one LUTs.The architectures of logic cells are different and usually its the number of LUTs not the number of registers that is bottleneck for FPGA. From this perspective, it could be stated that one Virtex-5 slice can be in theory substituted by 8 Virtex-4 slices or 4 ALMs.

2.4 SoC FPGA

SoC FPGA devices integrate both processor and FPGA architectures into a single device. Melding the two technologies provides a variety of benefits including higher integration, lower power, smaller board size, and higher bandwidth communication between the processor and FPGA. Best-in-class devices exploit the unique advantages of a merged processor and FPGA system while retaining the benefits of a stand-alone processor and FPGA approach [22].

As the signals between the processor and the FPGA now reside on the same silicon, communication between the two consumes substantially less power compared to using separate chips. The integration of thousands of internal connections between the processor and the FPGA leads to substantially higher bandwidth and lower latency compared to a two-chip solution.

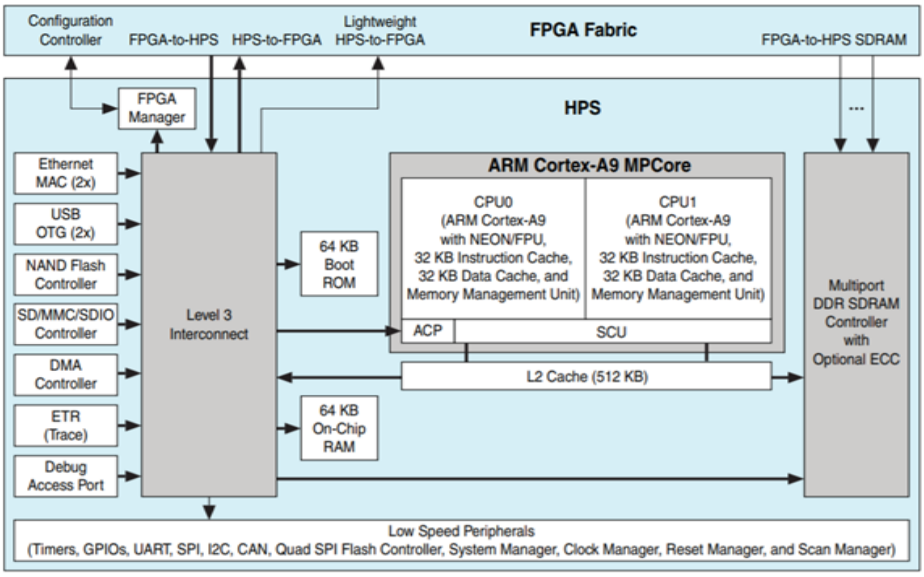


Figure 2.4. Cyclone V SoC FPGA - HPS with Dual Core ARM Cortex A9 MPCore Processor interconnect to FPGA Fabric [2].

Figure 2.4 shows the architecture of Altera Cyclone V, a SoC FPGA with HPS. The HPS consists of ARM Cortex-A9 MPCore processor, a rich set of peripherals and a shared multiport SDRAM memory controller. There are three HPS-FPGA AXI Bridges which support Advanced Micro-controller Bus Architecture (AMBA) Advanced extensible Interface (AXI) specifications: (a) FPGA to HPS AXI bridge is a high-performance bus supporting 32-,64- and 128-bit data widths allows FPGA fabric to be the master to slaves on HPS. (b) HPS to FPGA AXI Bridge is a high performance 32-, 64- and 128-bit data widths that allow HPS to be master to FPGA fabric slaves. (c) Lightweight HPS to FPGA is a low performance 32-bit width AXI bridge where HPS is the master [2].

CHAPTER 3

HIGH LEVEL SYNTHESIS AND OPENCL

3.1 High Level Synthesis (HLS)

High level synthesis, is an automated design process that converts untimed behavioral description of an algorithm in e.g., ANSI-C or C++, OpenCL to digital hardware that efficiently implements the required function. The complete HLS process is divided into multipe steps which includes syntax parsing, technology independent optimizations, allocation, scheduling and binding to create a Register Transfer Level (RTL) description in any Hardware Description Language (HDL), which can then in turn be synthesized to a gate-level netlist using a logic synthesis tool.

Due to the need to increase the design productivity and decrease in the time to market, HLS has finally become mainstream in most VLSI design companies which allows the design teams to meet their tight schedules. The main advantage of HLS is the increase in design productivity as less details are needed (only the functionality of the circuits needs to be specified), faster to design and verification leading to less bugs and easier to maintain the source code. Most of the systems today are heterogeneous systems. FPGA vendors have also released their own configurable SoC (CSoC) devices, e.g., Alteras Cyclone V SoC and Xilinxs Zynq FPGA and Intels recent announcement to acquire Altera is mainly based on the benefits of the tight integration of the x86 processors with FPGAs. As HLS raises the abstraction level of designing, it is easier to program all the units on heterogeneous system using same High level language.

One more advantage of raising the abstraction level is that, using HLS allows the generation of several micro architectures with different performance trade-off without having to modify the entire behavioral code. This is normally done by setting different synthesis options like program attributes or pragma.Therefore, HLS supports exploration of design space at Micro-Architectural Level.

3.1.1 High Level Synthesis Process

Figure 3.1 shows the complete HLS process starting from behavioral description. HLS tools use technology libraries of the target ASIC or FPGA and follows the three aforementioned steps: Allocation, Scheduling and Binding.

1. Allocation : In this step, the behavioral description is analyzed and the necessary hardware resources are determined. This step determines the Functional Units (FUs) required to realize the application. By default most of the HLS tools will try to maximize the parallelism as much as possible and hence allocate as many FUs as possible in the same cycle unless dependencies.

2. Scheduling: Determines in which clock step each the operations should be executed such that no precedence constraint is violated. Scheduling partitions the algorithm in control steps that are used to define the states in the FSM. Many scheduling algorithms have been proposed in HLS [23]. There are two most basic scheduling algorithms, As soon as possible (ASAP) and As late as possible (ALAP). ASAP maps operations to their earliest possible start time while ALAP maps operations to the latest possible start time, without any precedence violation.

3. Binding: In resource allocation each operation is assigned to a specific FU and each variable is mapped to a register. HLS tools are designed such that the units are reused to minimize logic requirement.

The typical hardware architecture generated after HLS contains two main paths: A control path which is a Finite State Machine (FSM) and a Data path, which includes hardware components, multiplexers, registers. Based on this design model the RTL is generated and described in any HDL languages like VHDL or Verilog.

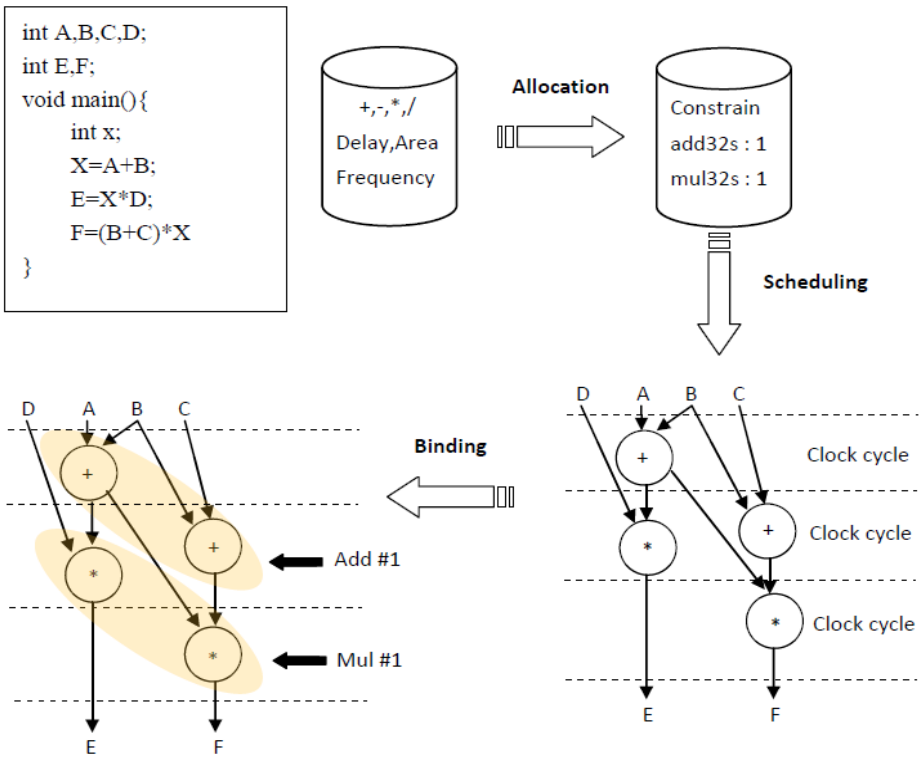


Figure 3.1. High Level Synthesis Process [3].

3.2 Open Computing Language (OpenCL)

OpenCL (Open Computing Language) is a standard framework from Khronos Group [24], allows parallel programming of heterogeneous systems that can include CPUs, GPUs, DSPs, and FPGAs. OpenCL programming language is based on C99 and C++11. It supports programming the devices on the heterogeneous platform and also the Application Programming Interface (API) to control the communicate between the compute devices. OpenCL provides a standard interface and API functions to support parallel computing using task and data based parallelism. In this section, we provide various concepts and technology background required to understand OpenCL Programming Model [3].

OpenCL Architecture is constructed with hierarchy of hardware units, consistent Execution Model and Memory Model which supports Parallel code Execution on heterogeneous systems.

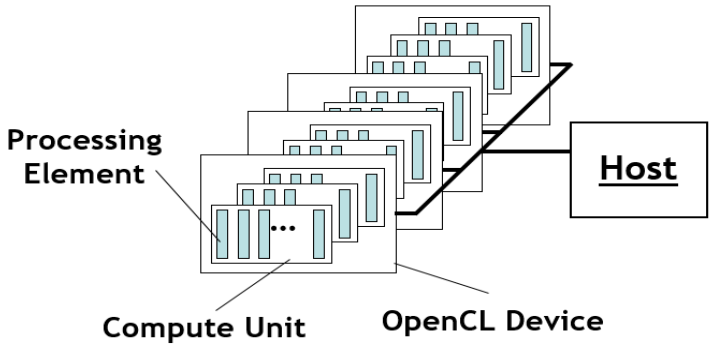


Figure 3.2. OpenCL Platform Model [3].

3.2.1 OpenCL Hardware Platform

OpenCL views a computing system as a number of computing units controlled by a Host Processor.The OpenCL hardware is layered to different abstraction layers (see Figure 3.2). At the top layer lies the CPU based host processor which is connected to one or more OpenCL Devices (For instance CPU, GPU or FPGA). Each OpenCL device consists of one or more Compute units (CUs), which are further divided into one or more Processing Elements (PEs) [3].

Computations are done within the processing elements. Multiple PEs, CUs and OpenCL devices can work in parallel. The host manages the workload division and data communication between the units using OpenCL API.

3.2.2 OpenCL Execution Model

The OpenCL applications have two execution units, the Kernel program and Host Program. The OpenCL host program executes on the Host. The host program uses OpenCL API functions to query the status of the Compute devices and manage the workloads across the Compute units. Whereas, the kernel program is executed on the Processing Elements(PEs). The kernel programs execute the computational part of the OpenCL application.

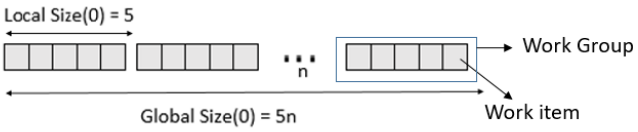
The entire data space which needs to be executed is termed as NDRange, where N denotes the number of dimensions of input data, which can be one, two or three. The NDRange is decomposed into Work-groups and Work-items (see Figure 3.3). Each independent element of execution in the entire workload is called a work-item. A set of work-items are grouped to into a Work-Group [4]. The decomposition is indicated by the Dimension(N), Global size and Local size variables. The Global size is defined as the total number of global work items in the NDRange for given dimension. The local size is defined as the number of local work-items in a work-group for given dimension. Figure 3.3 which illustrates the top view decomposition of a 1-Dimensional and 2-Dimensional NDRange. Work-Groups can be executed in parallel and distributed among the compute units. All the work items belonging to the same Work-group are computed by different Processing elements (PEs) of the same Compute unit.

3.2.3 OpenCL Memory Model

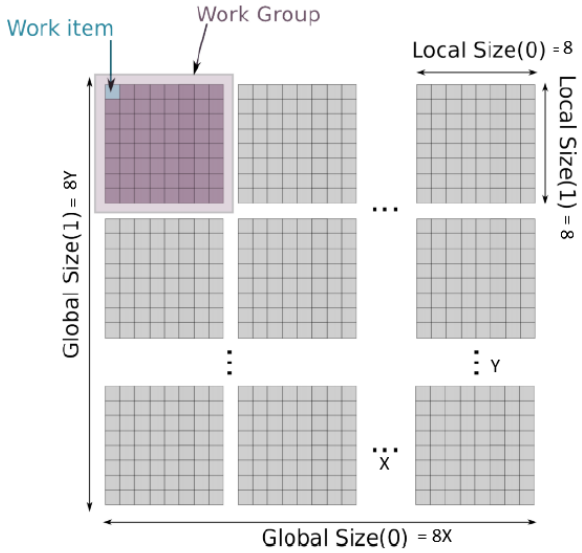
The OpenCL memory hierarchy is structured to support data sharing and synchronization of the work items. The OpenCL memory model consists of Global Memory, Local Memory, Private Memory and Host Memory (shown in Figure 3.4)

1. Host Memory: The memory directly available for the Host. Other computing units cannot access the host memory.

2. Global Memory: Global Memory can be accessed by all the compute units (CUs), processing elements (PEs) and the host. Therefore, it is shared by all work-items and work-groups. Global Memory Latency can be high so it is preferred to cache global memory to local memory when possible. The Part of the global memory which is constant throughout the execution is referred to as Constant Memory, Constant variables are declared here as it allows read only accesses for Compute units.



(a) 1 Dimension



(b) 2 Dimensional

Figure 3.3. (a) Example of Work-group and Work-item division for 1 dimensional. (b)Example of Work-group and Work-item division for 2 dimensional. [4]

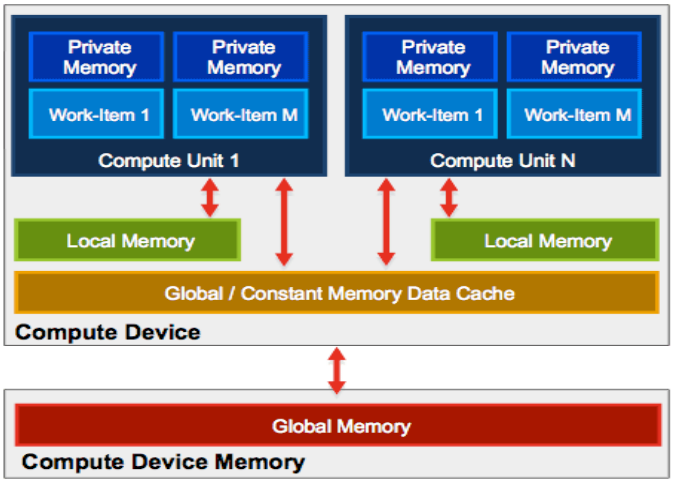


Figure 3.4. OpenCL Memory Model [4].

3. Local Memory: Each compute unit has a unique Local memory and cannot be accessed by other CUs. This memory region is local to the work group and can be shared by the work-items in the same workgroup. Variables defined in local memory are not visible to any other work groups.

4. Private Memory: Each Work-item has a private memory and is not visible for other work-items.

It is important that the read and write operations to Local and Global memory must be synchronized as they are shared memory. This is done by memory synchronization functions like Barriers and Memory fences [3].

3.2.4 OpenCL programming

For most of the parallel applications, the sequential part is executed in the host program and parallel computation parts are shared by the compute units which are programmed using the kernel programs. In addition, the host program should manage and coordinate the compute units which work in parallel. For this OpenCL API(Application Programming Interface) functions are used [25]. The OpenCL API functions allows the user to create and define objects like Context, Command Queue, and Data Buffers. The context defines the environment within which the kernel executes. A command queue is defined for each OpenCL Device. These queues enable the host to give commands and interact with the device. Buffers are used to transfer data between host and OpenCL device in the run time.

CHAPTER 4

HARDWARE ACCELERATION ON SOC-FPGA USING OPENCL

In the previous chapters we briefly discussed about SoC-FPGA, High level Synthesis Tools(HLS) and OpenCL Programming. In this chapter, we will discuss how using these technologies we can obtain acceleration on FPGA. Initially, we present the system description and methodology used to accelerate computations on FPGA. Then, we introduce the set of OpenCL benchmarks developed to study the acceleration behavior and factors affecting acceleration. Finally, we present the results obtained and observations.

4.1 System Description

4.1.1 System Hardware

All the experiments were made on Terasic DE1-SoC board [26], which consists of a hardware design platform built around the Altera SoC FPGA Cyclone V. Cyclone V [27] integrates an ARM-based Hard Processor System (HPS) consisting of processor, peripherals and memory interfaces tied seamlessly with the FPGA fabric using a high-bandwidth interconnect. This interface enables communication between the ARM processor and FPGA.

We used Linux terminal on the computer to interact with the FPGA Platform. The commands to the FPGA board can be given by user from the Linux terminal on the computer. The system is incorporated with a micro SD card installed with Linux image which enables to runs Linux commands on the FPGA board. An USB-UART cable is used to communication the Board and Linux host on the Computer. We used LAN port on the Board to transfer files between the Board and the Computer. The system is depicted in Figure 4.1

4.1.2 System Software

We used the following software to compile OpenCL programs and synthesize on the FPGA fabric:

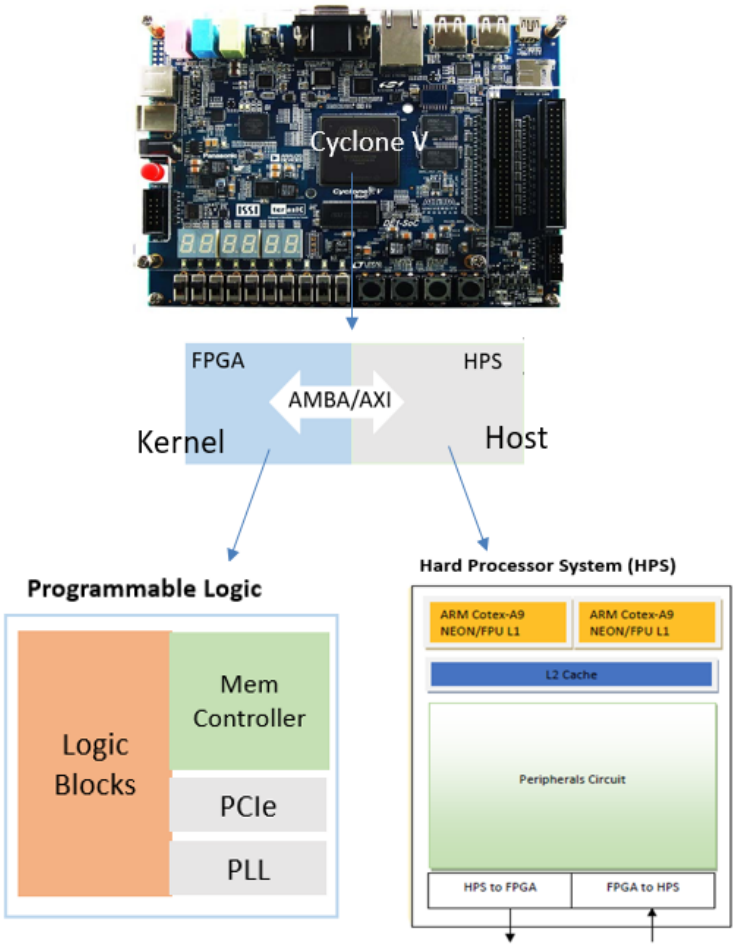


Figure 4.1. System Hardware Terasic DE1-SoC with Altera Cyclone V

1. Altera FPGA SDK for OpenCL (AOCL)

   Altera FPGA OpenCL SDK (Software Development Kit) is a HLS tool [5], which involves a complex flow to convert high level OpenCL code to hardware configuration files. It also allows functional verification by emulating OpenCL accelerator code on an x86-based host . The tool provides an optimization report with information about dependencies in the pipeline, memory stalls and timing information of the design. This report gives insights of the synthesized design which helps to modify the design for performance.

Compilation, OpenCL to HDL, emulation and optimization report generation processes may just take a few seconds. Whereas, the full flow synthesis process to generate Hardware configuration file (.aocx file) from OpenCL may take 1-5 hours depending on the design complexity. The end file from synthesis process with .aocx extension is used to configure the FPGA.

2. Altera Quartus II

   Altera Quartus tool is used to configure the target FPGA device from HDL (Hardware Description Language) provided by the user. This tool allows compiling of the designs, performing timing analysis and configuring the target device. This tool is required by AOCL compiler to generate a synthesized .aocx file

3. Altera SoC Embedded Design Suite

   The Altera SoC FPGA Embedded Development Suite (SoC EDS) is a tool for embedded software development on Intel SoC FPGAs. We used this tool to compile the OpenCL host program and generate an ARM executable.This executable file is executed by the Embedded ARM processor on the system.

4.1.3 System Memory Architecture

The system consists to three types of memory(see Figure 4.2), each are described below

1. Global Memory

   Global Memory is off-chip DDR (Double Data Rate) memory. The data communication between host and kernel can be done only through the global memory. Therefore, all the host to kernel interface happens at global memory. It offers high capacity and latency is high. Global memory is visible to all work groups.

2. Local Memory

   Local memory is on-chip memory, it lies on the same chip as the HPS and FPGA. It

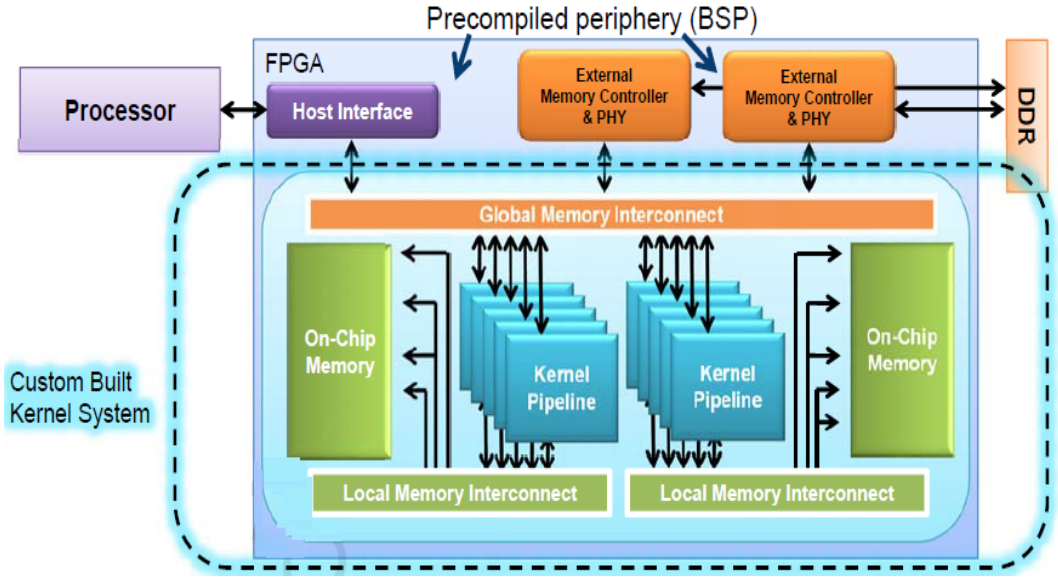


Figure 4.2. System Memory Architecture: Global, Local and Private Memory [5].

offers higher bandwidth and lower latency compared to global memory. Local memory is shared between work-items of same work-group.

3. Private Memory

   Private memory uses registers and block RAM on the FPGA. It offers performance at the cost of area. Private memory is unique to each work item.

4.2 Programming the System

Most of the applications have two types of code segments. The segments which needs be executed sequentially and the segments which can be executed in parallel. Sequential segments mostly involve data transfer instructions and set of instruction with dependencies. Whereas, the segments with parallelism involve loop instructions with less or no dependency, Single instruction multiple data computations etc.

Typically, sequential segments are performance efficient on general purpose processors (i.e. CPU) with pipelining, out of order execution, Memory hierarchy etc. Whereas, the

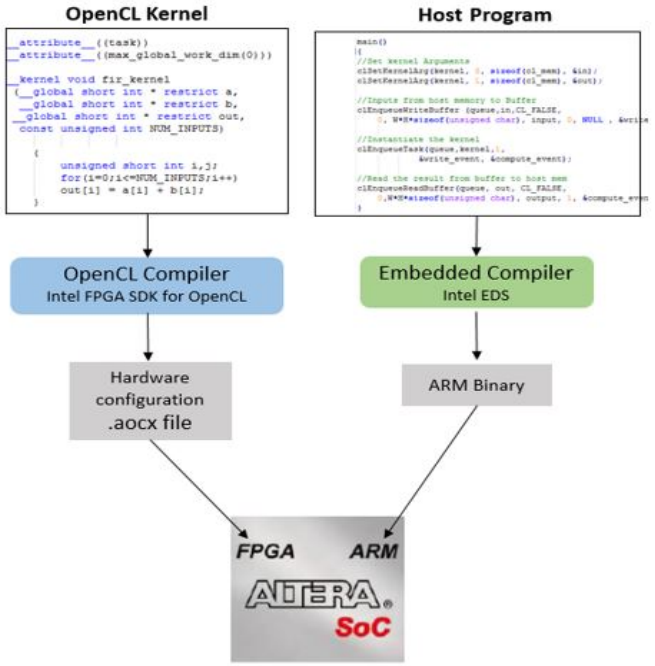


Figure 4.3. Overview of configuring the system with OpenCL code.

Parallel segments are performance efficient on the platforms which support parallel processing like Multicores, GPUs or FPGAs. Not only parallel tasks, FPGA can achieve performance on computationally intensive sequential tasks as well, by developing a dedicated hardware to compute these tasks.

SoC FPGA being a heterogeneous platform, provides benefits of both processor system (ARM) and re-configurable fabric (FPGA) for performance. The sequential segments can be executed by ARM processor, whereas the data parallel segments and computationally intensive segments can be executed by dedicated hardware designs on the FPGA. Here, the challenge is to establish a communication interface between the HPS and FPGA. Moreover, we need a common language to program both the systems and the interface.

Therefore, to meet these requirements OpenCL programming language is developed [24]. OpenCL supports development of programs that execute across heterogeneous units on a platform. OpenCL is widely used for parallel processing as it allows to configure the communication interface between units for both task and data based parallelism. Most of the

recent hardware devices and compiler tools from different vendors support OpenCL execution.

4.2.1 Host and Kernel

An OpenCL application views a computing system as combination of Host Processor (typically a CPU) and a set of accelerating OpenCL devices which can work in parallel, they are termed as Kernels.The Host device is responsible for issuing functions to the compute units and the control of data communication interface.

In this case, the ARM processor is the Host Processor and FPGA is the Kernel device. Each OpenCL application has two set of programs, the Host program and OpenCL Kernel program to program the host and kernels respectively. It is possible that the Kernel Device(FPGA) can have multiple compute units working in parallel. The Kernel code can be compiled, emulated and synthesized using Altera FPGA OpenCL SDK. Whereas, the host program can be compiled and host executable is generated by Altera SoC Embedded Design Suite. The flow of Programming system using OpenCL host and kernel programs is shown in Figure 4.3.

4.2.2 Host Program and Device Kernel Development Flow

The Host program is an OpenCL program with C/C++ based Library functions, compiled by Embedded design suite to generate ARM executable file. Kernel code is in standard OpenCL and compiled by Altera OpenCL FPGA SDK to generate hardware configuration file (.aocx), RTL files, HTML reports, synthesis reports and scripts etc.

Figure 4.4 shows the development flow to develop kernel for an application. These stages involve functional verification, optimization and performance improvement of kernel. Each stage provides different information about the design which help in optimizing the hardware kernel. Each step in the flow is described below:

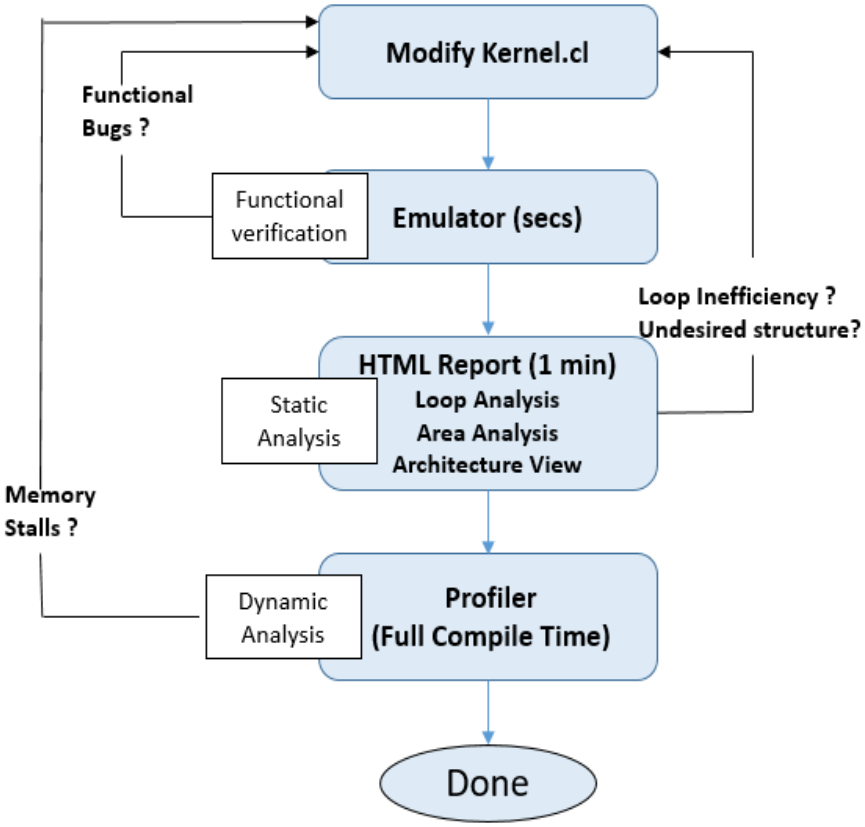


Figure 4.4. Kernel function development Flow.

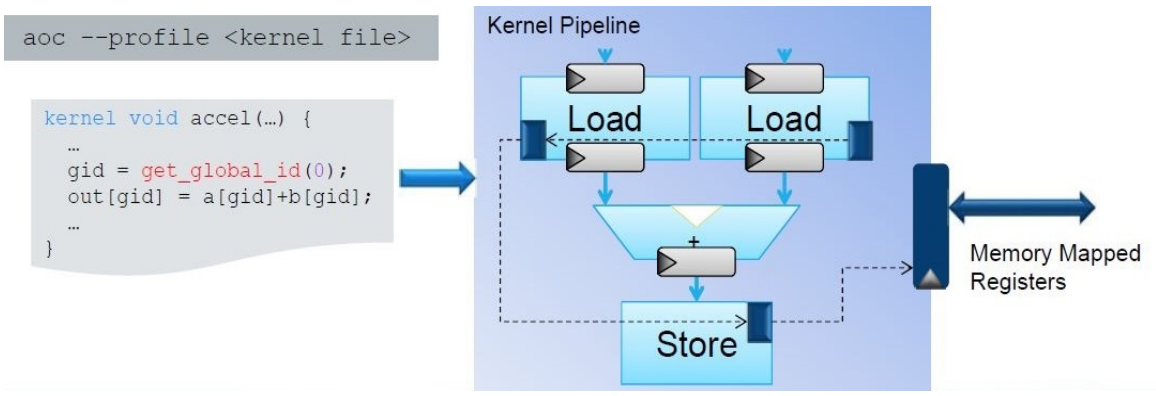


Figure 4.5. Analysis with Profiler, Kernel incorporated with Flip-flops at Load and Store Transactions [5].

1. Functional Verification: Emulation and Debugging

   This step is used to functionally verify and debug the host and kernel programs in a short time before starting the full compile flow. Using AOCL Emulator a .aocx file is generated within seconds, which emulates the functionality of the kernel. This .aocx file can execute on x86-64 windows or Linux host. To support debugging print statements can be used in kernel and there are debugging tools like Intel Code Builder which allow to set break points, read intermediate variables etc. Emulation resembles the functionally but does not give correct execution times. In fact, it might execute at slower speed than on actual optimized Hardware.

2. Static Analysis: Optimization Report Analysis

   The compiler automatically generates a HTML report file, which gives architectural overview of the system that will be synthesized. It consists of loop Analysis, Area Report and Architectural view of the system. The information from the report can be used to modify the kernel code for better performance.

   • Loop Analysis Report:

   This report shows the status of each loop in the application. It provides if loop unroll status, pipeline status and Initiation Interval (II) of each pipeline loop. Initiation Interval is defined as the number of cycles for which each new data enters into the pipeline. Any bottleneck due to data dependency or memory stalls can be identified.

   • Area Analysis Report

   The compiler provides a resource utilization report with hierarchical information. That is, the amount of resources utilized by each line in source code and each block in source code. Therefore, allows us to depict which part of the code is using most resources (like LUTs, Memory etc.). Generally, complex data operations, float point operations consume more logic resources and unrolled loops consume more memory bandwidth.

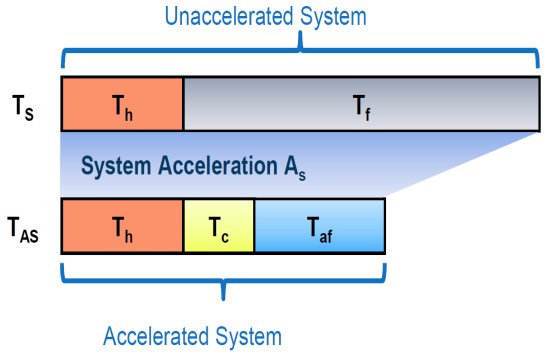


Figure 4.6. Time line of Unaccelerated system vs Accelerated System [5].

- System Viewer

  This part of the report displays the blocks in the system and the memory access implementation. It presents information about all Load/store stalls and latency for both global and local memories.

3. Dynamic Analysis: Profiler

   This step involves runtime analysis of kernel performance. In this process additional performance counters are added at each load and store in the design as shown in Figure 4.5. These counters collect kernel performance data and send to host program. This information can be viewed via GUI. Profiling is generally used to identify which channel in the pipeline is causing the memory stalls.

4.3 Optimization of Design for Acceleration

In this section, we discuss the methods and architectures to improve data processing efficiency on FPGA and different optimization metrics which affect the performance. As per the timing diagram in Figure 4.6 and timing notations in Table 4.1, the major factors through which acceleration can be achieved are discussed.

1. Reduce T_{af}: Increase the number of parallel operations

   This can be achieved by increasing the number of parallel operations.

Table 4.1. Timing notations.

Timing Notation	Description
T_s	Total Time spent on an unaccelerated system
T_h	Time spent on an unaccelerated system for the host part
T_f	Time spent in an unaccelerated system for the Un-accelerated function
T_{AS}	Total Time spent on accelerated system
T_C	Time spent on communication between host and accelerator
T_{AF}	Time spent on an accelerated system for the accelerated function
A_S	System Acceleration = T_s/T_{AS}

- Data level parallelism:

  In applications where the data can be partitioned and computed independently, data level parallelism can be deployed. This is done by replicating Compute units, vectorising data processing units on the kernel and launching them in parallel. Similarly, loop iterations can be paralleled by loop unrolling.

- Instruction level parallelism:

  Instruction level parallelism can be achieved by pipelining the kernel application. Dependencies between the instructions can be handled by data feedbacks.

- Task level parallelism:

  A set of independent tasks can be run in parallel on different kernels. Task level parallelism is achieved by dividing a big problem into separate kernel devices and run them in parallel.

2. Reduce T_c: Reduce Communication Overhead:

   To reduce communication latency between host and kernel, the following practices can be implemented to reduce global memory accesses. The system allows host to kernel communications only through global memory. Due to high global memory latency, global memory accesses from kernel must be reduced when possible.

   - By moving frequently used data to local memory or On-chip memory.
   - Using Channels and pipes to communicate between kernels, instead of global

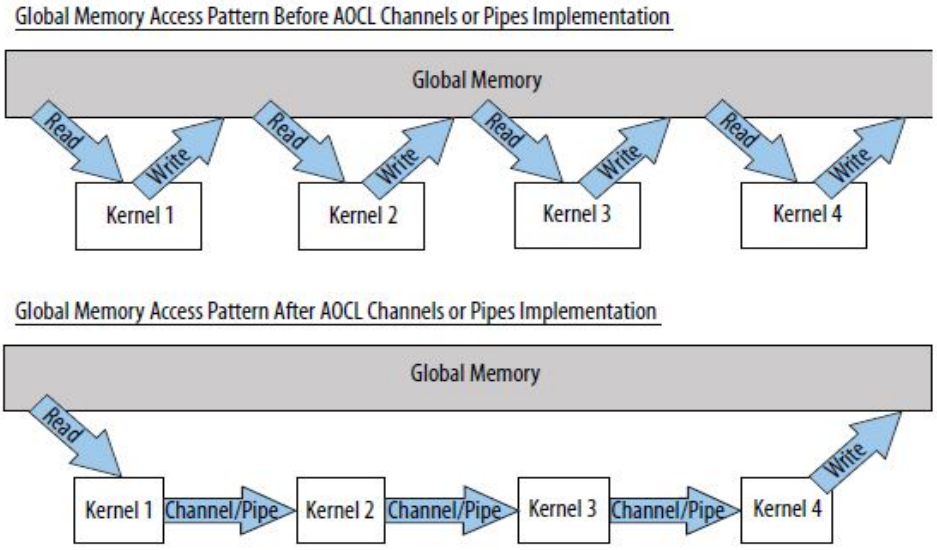


Figure 4.7. Kernel to kernel communication : using Global Memory vs Channel/Pipes [5].

memory to pass the data in between. Figure 4.7 compares kernel-kernel communication through Global Memory vs Channels/Pipes.

4.3.1 Optimization Architectures

The Altera SDK for OpenCL (AOCL) compiler offers two different data processing architectures for the kernels, they are Single Work Item Execution and NDRange Kernel Execution [28]. Depending on the application one or a combination of both can be chosen for the kernel design on FPGA. At the early stage of design planning one should decide whether to construct the OpenCL kernel as a Single work item kernel or an NDRange Kernel. Both architectures are described in detail below.

1. Single Work Item Kernels

   • The entire kernel application is executed as a Single thread. In this Single work item execution mode, the kernel consists of one compute unit which executes work items sequentially one after another. When a kernel is defined with an attribute 'task' the

compiler will automatically synthesis it as a single work item execution kernel.

• In these designs, the throughput is achieved by a method called Loop Pipelining, shown in Figure 4.8. In this method, compiler will infer pipelined parallel execution across the loop iterations. That is, the loop structure is pipelined and multiple loop iterations will be executed simultaneously in the pipeline. Figure 4.9 shows an example of OpenCL kernel realized as a Single Work Item Kernel, it depicts that when work item 1 is at add stage, work item 2 is at load stage in pipeline.

• For Single task kernels, the throughput mainly depends on the factor called Initiation Interval (II). It is defined as the number of cycles for which every new data launches into the pipeline. It determines the launch frequency of loop iterations. Therefore, minimizing the II is the key to single work item kernel performance (refer equation 4.1). Best performance is obtained when new data is launched every cycle (i.e., II=1). Higher II can be reduced by relaxing the loop carried dependencies with methods like converting nested loops to single loop, using local variables instead of global variables etc.

$$Exec\_time = ((Num\_iterations * II) + (Loop\_latency)) * Time\_period \tag{4.1}$$

• The data dependencies within the successive loop iterations will be resolved by incorporating data feed backs on the FPGA hardware as illustrated in Figure 4.9 . Here, each iteration is dependent on C[i] value written in the previous iteration. Therefore, the computed C[i] is shared through a feedback to the next iteration. So the next iteration can be launched as soon as the dependency is cleared.

• This mode is ideal when there is dependency between the work items and no data level parallelism is possible between the work Items. Therefore, in cases where it is difficult to partition data among parallel work items or if data must be shared between

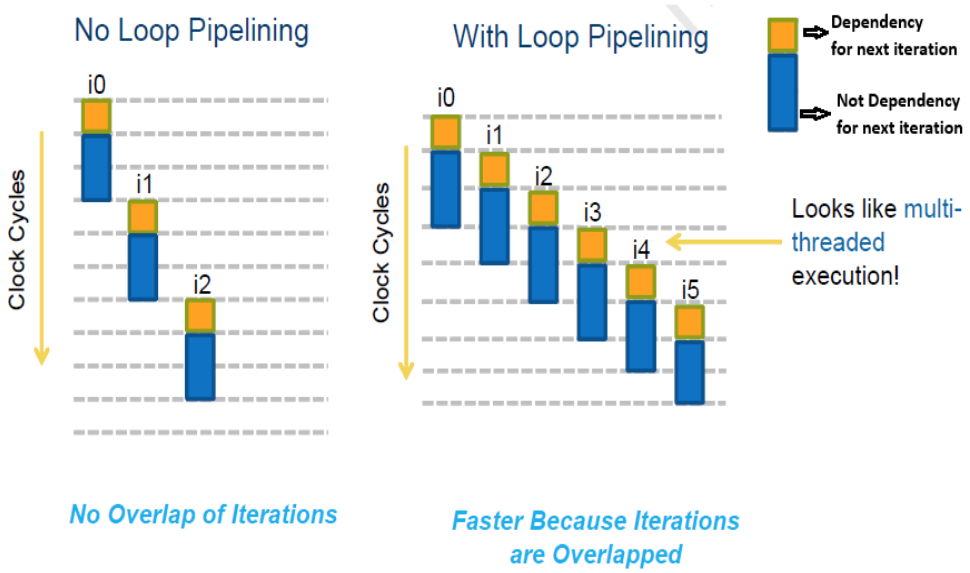


Figure 4.8. Iterations in No loop Pipelining vs Loop Pipelining [6]

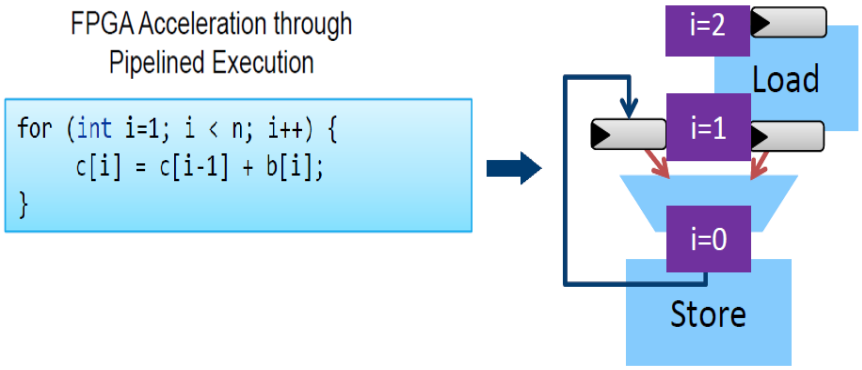


Figure 4.9. Single work item implementation showing a data feedback to handle dependency [6].

the work items, Single work item execution provides better throughput.

• For example, FIR filter, Interpolation Filter, Decimation filter are good to implement as Single Work Item kernels because the new outputs depend on previous inputs and output data.

2. NDRange Kernels

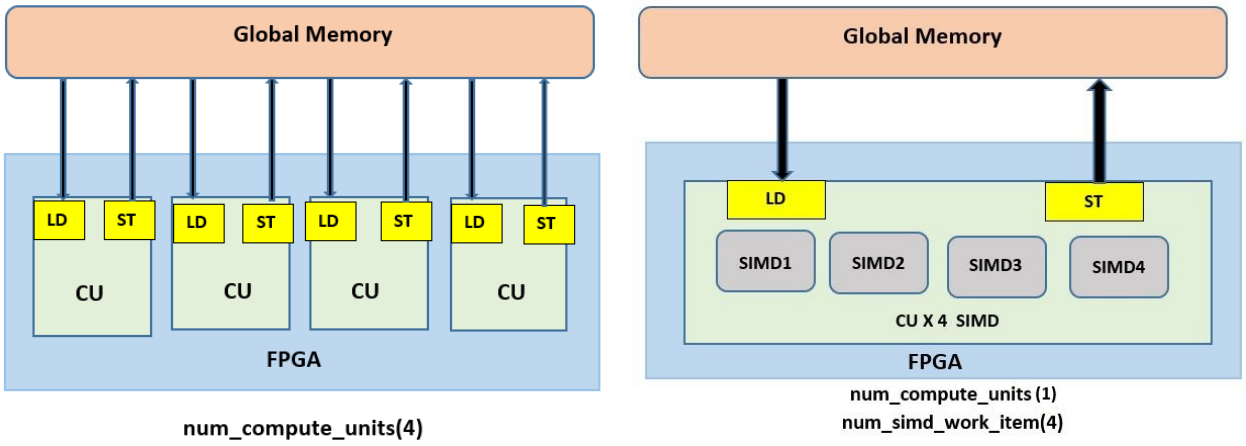
• In NDRange Kernel, Data processing efficiency is achieved by implementing data level parallelism. Hardware is replicated to allow parallel execution. NDRange kernel can have a number of compute units (CUs) and a number of Single Instruction Multiple Data processing units (SIMDs), which simultaneously work on different work groups or work items in parallel.

• Achieves throughput by parallel processing at the cost of Memory Bandwidth and Logic utilization.

• An application can be implemented as an NDRange kernel, if the kernel program has no loop or memory dependencies between the work-items. For example, applications like AES Encryption, Matrix Multiplication etc .

• CUs work on different work-groups in parallel, whereas SIMDs work on different work-items which belong to same work group. In Compute unit duplication, the entire units of datapath, control path and memory access blocks are replicated. In contrast, for SIMD vectorization only the datapath of the compute unit is duplicated and the control logic is shared.

• Figure 4.10a and Figure 4.10b depicts the difference between compute unit replication and SIMD vectorization. Though both offer throughput they provide different efficiency. Multiple compute units consume more logic utilization and create undesired memory access patterns as they work on data from different work groups. Whereas,



(a) (b)

Figure 4.10. (a) Kernel with 4 compute units (b) Kernel with 1 Compute unit and 4 SIMD units per Compute Unit.

SIMD vectorization allows to coalesce memory accesses and is area efficient.

• Therefore, in most cases SIMD vectorization is preferred over multiple compute units. In few cases, when work items have different control logic and cant share a single control path multiple compute units are used for parallelism.

4.3.2 Optimization Pragmas and Attributes

Adding the following attributes and pragmas to OpenCL Kernel code allows compiler to modify the kernel Micro-architecture accordingly [28]. Each has its own advantages and disadvantages over efficiency. By changing these attributes in different combinations, we can create different hardware configurations.These attributes affect the Logic utilization, memory Bandwidth, memory Access patterns and performance in different ways.

1. *num_compute_units*(N): This attribute duplicates N compute units on the kernel. This duplicates the control path and data Path. These CUs work in Parallel on different work groups, due to which it is possible that they have wider memory access patterns.

Table 4.2. System description of the benchmarks

Benchmark	Kernel	Pipeline status	II	Fmax (MHz)	Logic (%)	Highest Acceleration
Sobel	Single task	Yes	2	136	20	7.3
FIR	Single task	Yes	1	168	20	0.95
ADPCM	Single task	Yes	40	158	20	2.6
Decimation	Single task	yes	1	129	81	2.8
Interpolation	Single task	yes	1	125	28	2.8
AES	NDRange CU=2,SIMD=2	No	N/A	135	84	6.9

2. *num_simd_work*(N): This creates N number of SIMDs per each compute unit. The value of N is limited to 2, 4, 8, and 16. This duplicates only the data path, the control path is shared. SIMDs work on workitems of same workgroup. Therefore, memory access patterns can be coalesced.

3. *#pragma unroll* $< N >$: This Pragma unrolls the following loop N times. Increases the memory bandwidth requirement.

4. *max_work_group_size*(N): This attribute N determines the maximum work group size the kernel should execute.

5. *reqd_work_group_size*(x, y, z): This attribute determines the exact work group size the kernel should execute in three dimensions.

4.4 Acceleration Benchmarks and Results

4.4.1 OpenCL Benchmarks

We have developed six acceleration benchmarks including Sobel filter, FIR filter, ADPCM filter, Decimation filter, Interpolation and AES encryption in OpenCL. These benchmarks are developed from S2CBenchmark suite [29], which are created in SystemC. All the benchmarks are written in OpenCL with a C based OpenCL host program and a standard OpenCL

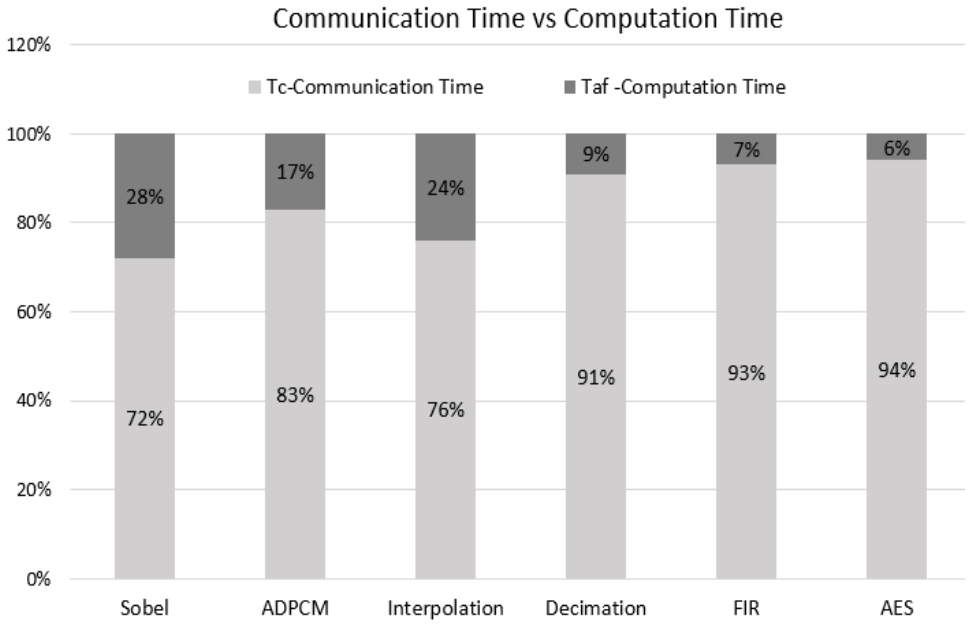


Figure 4.11. Communication vs Computation Time on Accelerated System (ARM+FPGA).

Kernel. The main goal of these benchmarks is to accelerate their execution on SoC FPGA systems. These benchmarks are portable to most of the Altera SoC-FPGA with few to no modifications. These benchmarks can be used to generate different hardware Micro-Architectures by changing the attributes and parameters.

- Sobel: Sobel is a 3X3 , edge detection image filter developed for 8-bit .bmp images
- FIR : A 10 Tap Finite Impulse Response Filter
- ADPCM : An Adaptive Differential Pulse Code Modulation encoder for 16 bit Pulse Code Modulation (PCM) samples.
- Decimation: A 5 Stage Decimation Filter.It consists of 5 Cascading FIR Filters, where the output of one filter is fed to the next Filter.
- Interpolation: A 4 Stage Interpolation Filter
- AES Encryption : A 16 bit data and 128 bit key AES Encryption

Table 4.3. Acceleration of AES by varying number of CU and SIMD attributes across different data sizes,work groups = N and work items = (Num Inputs)/N ,where N=2,4

workgroups	Compute units	SIMD units	256 Inputs	512 Inputs	1024 Inputs
2	1	1	2.5	4.6	6.6
2	1	2	2.7	5.4	6.6
2	1	4	2.4	5.4	6.9
2	2	1	4.0	4.6	6.9
2	2	2	2.6	5.2	6.9

(a) Number of workgroups = 2

workgroups	Compute units	SIMD units	256 Inputs	512 Inputs	1024 Inputs
4	1	1	0.7	2.6	6.2
4	1	2	1.0	2.7	4.6
4	1	4	3.8	5.5	5.7
4	2	1	1.3	2.1	5.7
4	2	2	1.2	2.0	6.5

(b) Number of workgroups = 4

We compiled each benchmark kernel using AOCL compiler and Host program using Altera EDS. We deployed each benchmark on Terasic DE1-SoC board which has Altera Cyclone V FPGA. The Benchmarks are designed such that, each Benchmark runs the application in two versions, first version is on the unaccelerated system (ARM) and second version on the accelerated System (ARM + FPGA). The corresponding Execution times in both the systems are recorded. The execution time on ARM+FPGA system with respect to the execution time on ARM processor alone is used to determine the System Acceleration obtained. Table 4.2 shows the system parameters for different Benchmarks. The following are the Timing are determined by the benchmarks

- arm_time = Computation time of application on ARM processor alone
- $write\_time$ = Time consumed for Input transfer from Host to Input Buffer
- $read\_time$ = Time consumed for Outputs transfer from Output Buffer to Host

- *api_time* = Time taken in-out data communication between Host(ARM) and Kernel(FPGA)
- *comp_time* = Computation time of application on FPGA kernel
- $hw\_time = api\_time + comp\_time$
- $acceleration = arm\_time/hw\_time$
- *api%* = (*api_time* / *hw_time*) *100

4.4.2 Experimental Results

For each benchmark, the Execution times from both unaccelerated execution (ARM) and Accelerated Execution (ARM+FPGA) and the speed up is determined for several Input data sizes. Therefore, the results show the acceleration on ARM+FPGA system with respect to ARM.

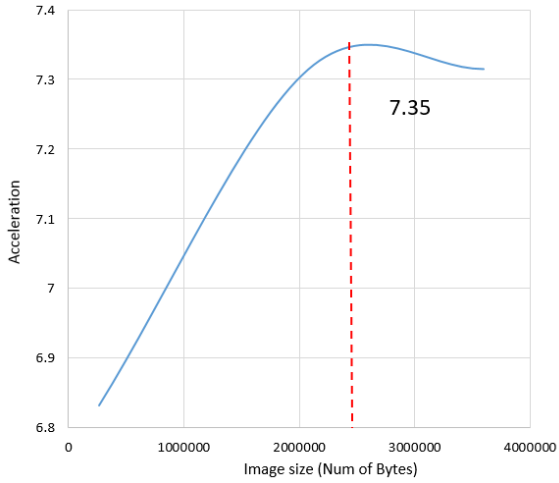
The plots in Figure 4.12 represents the trends in acceleration with respect to the Input data size. It can be observed that, initially the acceleration tends to increase with the input data size because with the growing computational complexity, the resulting computational acceleration overcomes the communication overhead between the host and kernel. In common for all the benchmarks, the acceleration tend to saturate after reaching certain input size. The computational acceleration freezes beyond a point as there is no immediate data is available for processing due to communication overhead and limited data buffer size between host and kernel.

The plot in Figure 4.11 depicts the ratio between the time taken for computation to the time take for communication between the host and kernel. For all the benchmarks, most of the execution time is spent on data transfer between the host (ARM) and the kernel (FPGA). The highest speed up for FIR filter is 0.95 (See Figure 4.12b), there is no acceleration achieved because of 93% of communication overhead.

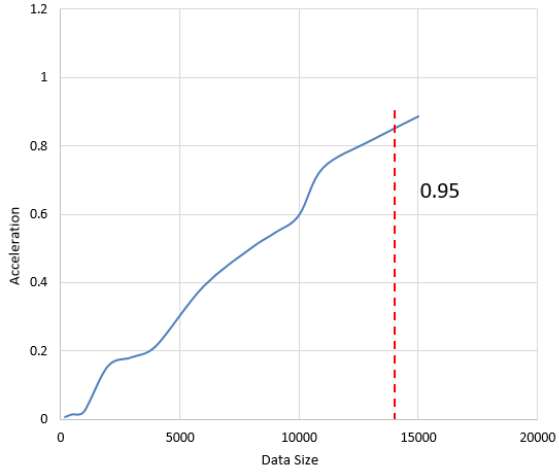
Table 4.3 presents the acceleration values obtained for different Micro-Architectural configurations of AES benchmark design. AES is designed as a NDRange Kernel in which data workgroups can be executed in parallel. By varying the attributes say number of CUs, SIMDs and Work-group size the performance of the design effected due to change in Data Efficiency, Memory access due to which the speed up varies.

4.4.3 Conclusions

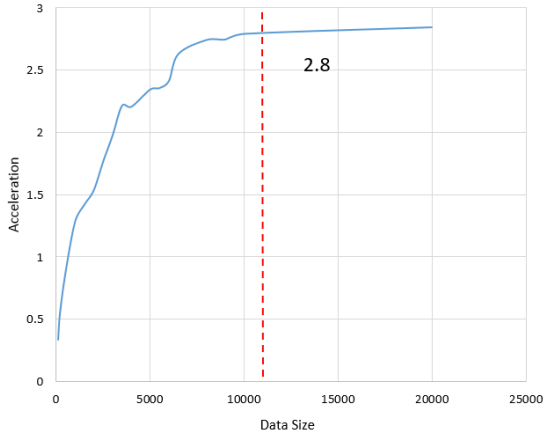
This Chapter presents the Architectures and Attributes which affect the performance of the system. We introduced few OpenCL Benchmarks which are mapped to Cyclone V SoC-FPGA. An average speed-up of 4 times is observed and most of the execution time is consumed for data transfer between host and kernel. The results show that at higher input data size the acceleration saturates, because of the Computational acceleration is limited by the communication overhead.



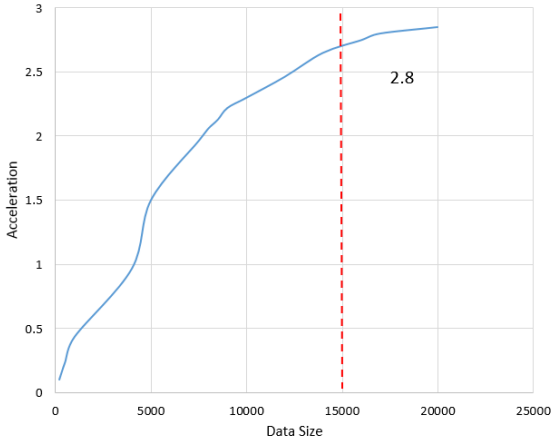
(a) Sobel



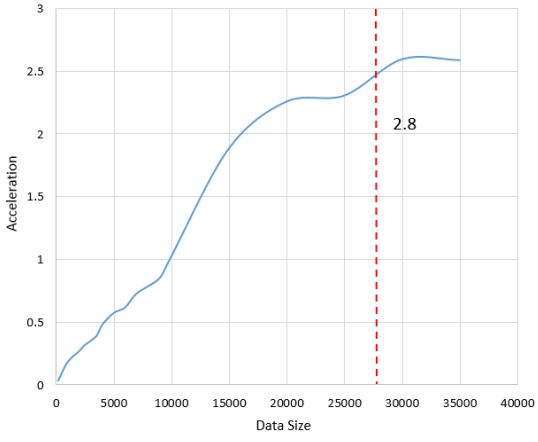
(b) FIR



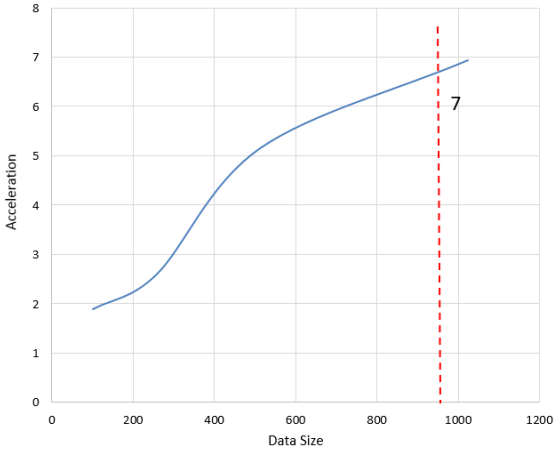
(c) Decimation



(d) Interpolation



(e) ADPCM



(f) AES,CU=2,SIMD=2

Figure 4.12. Plots of Acceleration vs Input Data Size for the benchmark suite.

CHAPTER 5

DESIGN SPACE EXPLORATION

The activity of exploring the design alternatives of a system prior to the design implementation is called Design Space Exploration (DSE). DSE refers to systematic analysis and pruning of unwanted design points based on parameters of interest [30]. It is used for many applications like rapid prototyping, trade off analysis and to find optimal design with respect to the parameters of interest etc. One of the advantage of VLSI design at behavioral level is that HLS allows to alter the Micro-Architectures without having to modify the behavioral description. This is done by setting different synthesis options like Attributes and Pragmas (Loop unroll factor, Number of Compute units, and Number of SIMD units etc.) in the code, as discussed in the previous chapter.

Each of these attributes directly affect the performance, resource utilization and memory access patterns of the design by changing the Micro-Architecture. Therefore, different combinations of these attributes, create a large design set with multiple configurations and varying performance. In Most of the cases, it is difficult to predict the right combination of attributes which provides the optimal solution because these attributes can interact with each other in unforeseen manners [31]. For example, increasing the number of compute units, which is duplicating processing units and running in parallel should increase the performance. However, this may not be true always as the high memory bandwidth requirement can degrade the performance as a result of memory stalls, such trade-offs offered by these attributes varies from application to application. Thus, it is not easy to anticipate the optimal design without testing many designs. In addition, the search space is extremely large and it is not practical to prototype entire solution space due to Compilation time constraint. This motivates to develop a faster heuristic approach to search the solutions space for a good solution, fast enough.

In this Chapter, we propose and analyze a fast and heuristic DSE method using Genetic Algorithm (GA) to search the solution space by evaluating the effect of these control attributes. Here we present the methodology of DSE using Exhaustive search and a Fast Heuristic algorithm, then compare the results of both approaches with well-defined quantitative metrics.

5.1 Design Space Exploration with Exhaustive Search

Exhaustive search DSE involves analyzing of all possible search combinations. It has the advantage that it is able to find the optimal solution and it is easy to implement. The main challenge is that the size of the design space is typically extremely large and groups exponentially with the number of exploration options.

5.1.1 Methodology

Figure 5.1 shows the steps for DSE methodology,each step is described below

1. Add Attributes

   For Each benchmark, we determined the relevant attributes (e.g., Loop Unroll factor, CUs etc.), which can affect the kernel performance. The OpenCL KerneL code is incorporated with these attributes and their corresponding values can be tuned to generate different kernel configurations.

2. Generate possible designs by varying the attributes

   We used scripts to tune the optimization parameters with appropriate values for each kernel. This step generates several kernel configurations with different combinations of optimization attributes. The number of designs generated for each benchmark and that could fit into the resources ranged between 50 to 200 .

3. Compile Each Design

   Each design generated in the previous step is synthesized using Altera OpenCL SDK (AOCL) compiler to extract the different design metrics (i.e. area, latency, delay). The synthesis time for one design typically takes 1 to 4 hours depending on the design complexity. So, the compilation of the entire solution set exhaustively for each small benchmark takes several days (typically 5-7).

   We use Kernel area and Execution time as metrics to compare the performance of the designs. These metrics are extracted for each design from the report files generated by the compiler tool. The Logic utilization is extracted from the Synthesis report and the timing is estimated form Latency, Initiation Interval (II) and Maximum Frequency (Fmax) using equation 4.1. The execution time also depends on the input size, so we assumed a fixed input data size for all kernel designs.

4. Plot Design Space for analysis

   Design Space is created by plotting all the designs with Area and execution time values as vertex co-ordinates. To illustrate, the DSE of Sobel benchmark performed for image size of 1024 by 1024 is shown in 5.2

5. Reference Pareto Optimal Front

   A good design is expected to have lower execution time along with lower logic utilization. In DSE, a design X is referred to as dominated by another design X* if and only if, X* is better than X with respect to all parameters. The Solutions which are not dominated by any other solutions in the Design Space are referred to as Non-dominated Solutions. It is important to note that a Design Space can have multiple Nondominated solutions. Pareto Optimal Solutions is the set of non-dominated solutions, for which no parameter can be improved without sacrificing at least one other Parameter. In this case, the Pareto front is the solution front which is closest to axes, as we expect area

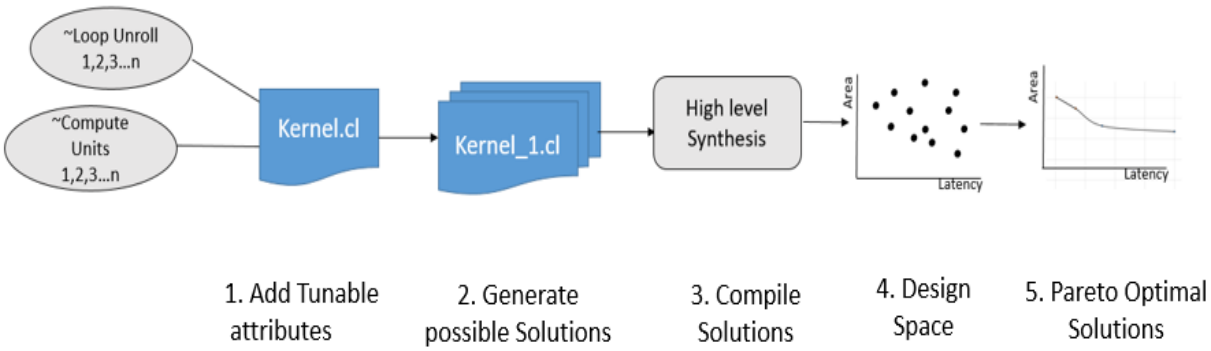


Figure 5.1. Steps for DSE

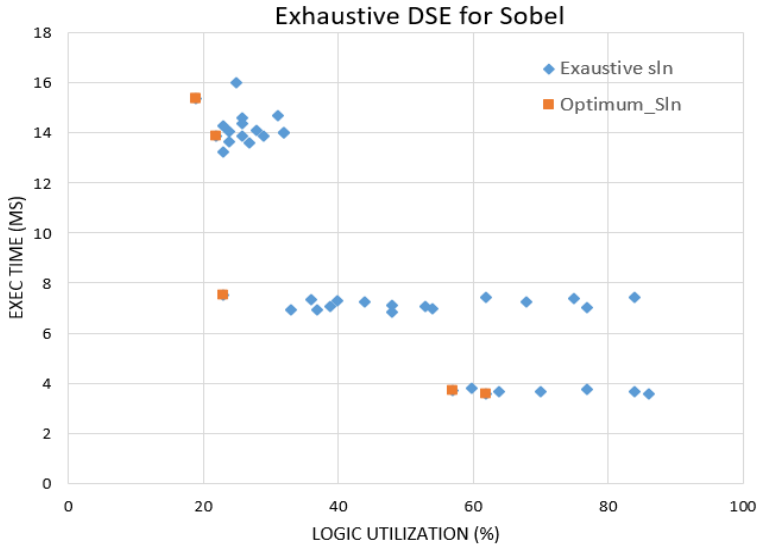


Figure 5.2. Exhaustive Design Space for Sobel, with Pareto optimal Solutions identified.

and execution time to be low. For each benchmark, we manually identified the Pareto Optimal Solutions. Figure 5.2 illustrates the Pareto Optimal Solution for Exhaustive DSE of Sobel benchmark.

5.2 Design Space Exploration with Genetic Algorithm

In the previous section, we discussed the use of and exhaustive search method to perform the DSE. Due to the exponential nature of the problem, faster heuristics are needed. To approach

this problem, we can automate DSE with an exploration software that can make decisions and come up with a good solution in comparatively less time. For this, we need a heuristic search algorithm, which makes decisions such that the successive iterations are directed towards the better solutions and also has a convergence criteria to minimize the number of iterations during exploration. Moreover, the algorithm must support exploration in spite of trade off between several attributes. To meet these requirements we used Genetic Algorithm (GA) which has ability to generate good enough solution fast enough. Genetic Algorithm applies the principle of natural selection and is formulated by bio-inspired operators such as mutation, crossover and selection. GA offers a good proportion of randomness and control logic to search the space. Although, GA starts randomized, it exploits historical information and direct the search to designs with better performance.

5.2.1 Methodology

The Genetic Algorithm takes a few control inputs from user such as Number of Mutations (M), Number of Parent Initiations (P), Count Value for convergence Criteria (N) and Area Cost Factor (a). The Genetic Algorithm graphical flow diagram is represented in Figure 5.3.

Each solution in the entire design space is represented with a unique array of Optimization Parameters (Loop unroll, SIMD etc.).This array is analogous to genetic set of an individual. A set of two solutions (parents solutions) are randomly chosen from the entire design space. A random cross over point is chosen and the parent attributes are combined accordingly. This is followed by M random mutations, the mutation locations are replaced by values from pre-defined knob values defined for each attribute. The new offspring is compiled for full synthesis flow to determine the cost function (5.1) using area and execution time of the design.

$$cost\ function = a * (Normalized\_Area) + b * (Normalized\_time) \tag{5.1}$$

The survival function is to discard the solution with high cost function. The better solutions with low cost function will continue to iterate around the genetic operations (Crossover and Mutations) to produce new generations till the convergence criteria is met. According to Convergence Criteria,the iterations are terminated when there is no decrease in the cost function for N consecutive iterations. At this point we consider the algorithm has converged to a solution. After GA reaches a convergence point, the algorithm is repeated for new set of parents chosen randomly. This loop continues for P different Parent sets and the then the termination criteria is met. The optimal solutions from different P iterations are compared and the best solution is returned.

The area cost factor is varied from 0 to 1 with a step value of 0.1.The set of solutions are plotted with area utilization and execution time as coordinates, the dominant solution front is determined and compared with Pareto Optimal Front.

5.3 Experimentation Results

4 benchmarks are used to test our proposed search method. In particular, Sobel, FIR, ADPCM and Interpolation filter. The other 2 benchmarks Decimation and AES are excluded due to the resource constraints on the board as not many of their design variations fit on the single FPGA. The design space is compiled for Terasic DE1 SoC which contains Altera Cyclone V SoC and the compilations are done using Altera OpenCL SDK (AOCL). The possible designs which are confined to the resources available on a single board are tested. The experiments are conducted in order to fully understand the effectiveness of our proposed fast heuristic method with respect to the Exhaustive search which leads to optimal solutions in all cases. To compare these two multi-objective function optimization methods, we used few standard metrics to quantify the Quality of results(QoR) of the heuristic with reference to the optimal and also compare the running time differences.

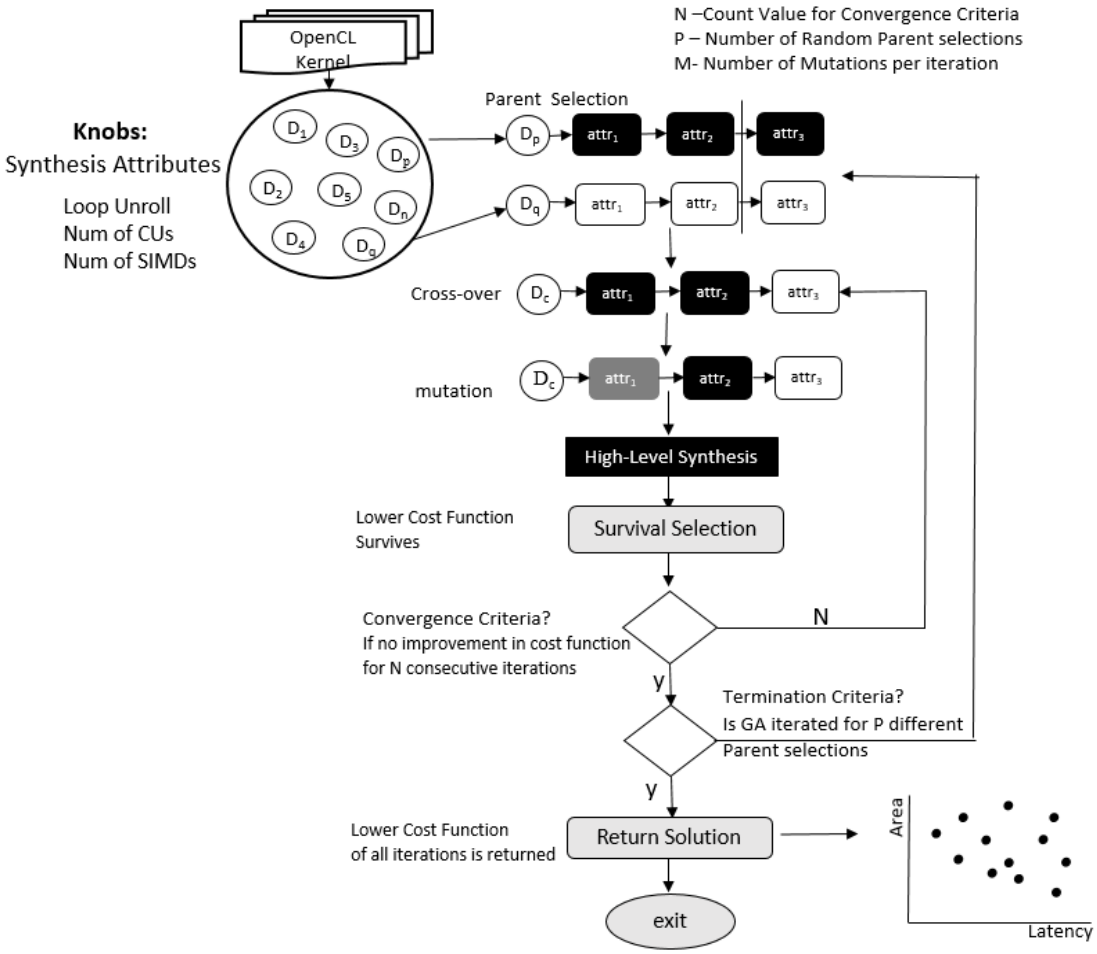


Figure 5.3. Flow of DSE using Genetic Algorithm.

5.3.1 Comparison of Results (Exhaustive search vs Genetic Algorithm)

To analyze the performance of the fast Heuristic method, we consider factors like closeness to the reference Pareto front, range of diverse solutions in trade off with the total compilation Time. There are few studies proposed on addressing the metrics to compare the approximations of trade-off surface in quantitative manner [32]. The metrics used for evaluating the exploration method are as follows:

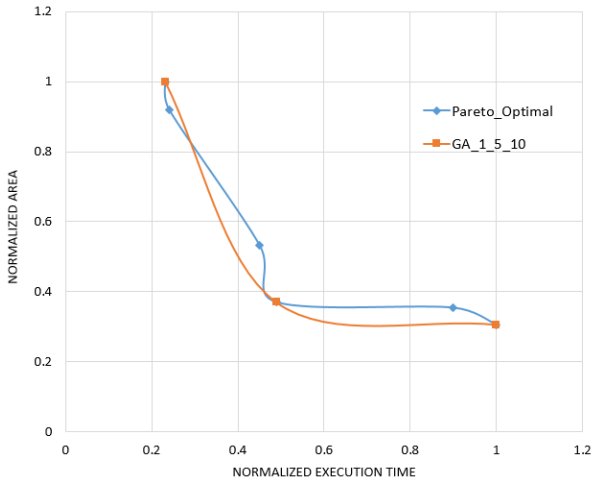
1. *Pareto Dominance*: This index is the fraction of the number of points common in the Pareto front being evaluated and the reference Pareto front. It quantifies the range of

diverse solutions found by the multi objective optimization method being evaluated. The higher the value, the better the Pareto set is.

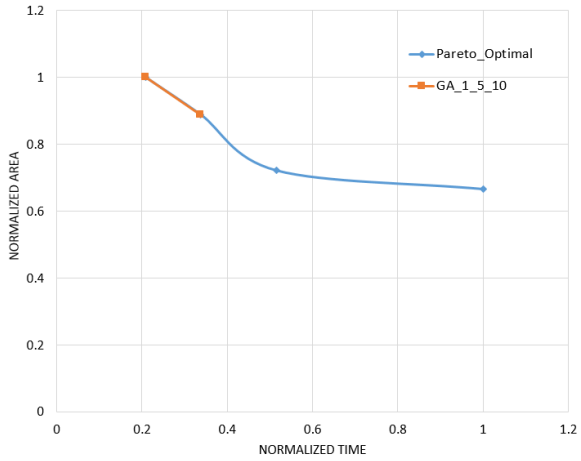
2. *Average Distance from Reference set*($ADRS$): This index is used to quantify how close the evaluated Pareto Front is to the reference front. It measures the average distance between the Pareto Optimal front(R) and the approximated Front(A). The lower the value, the better the Pareto set is.

The Plots in Figure 5.3, shows graphically the system exploration trade off curves, that is the comparison of fast heuristic method Pareto dominant front with the reference Pareto Optimal front from Exhaustive exploration. The designs are plotted with normalized area and normalized execution time as their coordinates. The effectiveness of the exploration method is determined by the closeness between the solution fronts(ADRS) and the number of common solutions (Dominance).

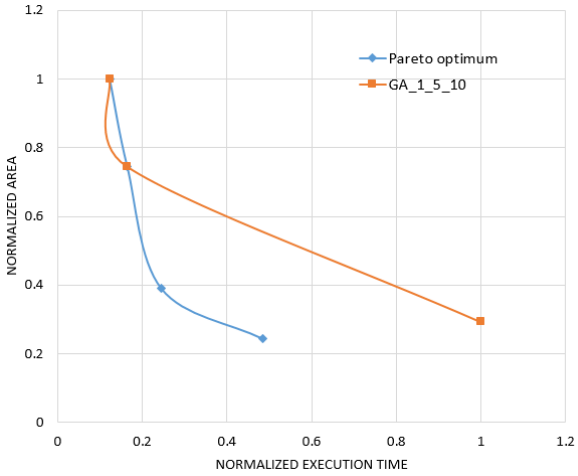
Table 5.1 presents the comparison metrics Dominance, ADRS and Speed up in the compilation time of Genetic algorithm with respect to Exhaustive search. The Genetic algorithm control parameters are varied to analyze the affect on the efficiency of the approach. In summary of Table 5.1, with average dominance of 0.7 the heuristic can determine about 70% of the optimal dominant solutions. The average ADRS of 0.2, that is the genetic algorithm solution can be within a 20% distance from the optimal solutions. The average Speed up in the compilation time is about 6 times for this benchmark suite. Thus, we can conclude that our method is effective in speeding up the DSE.



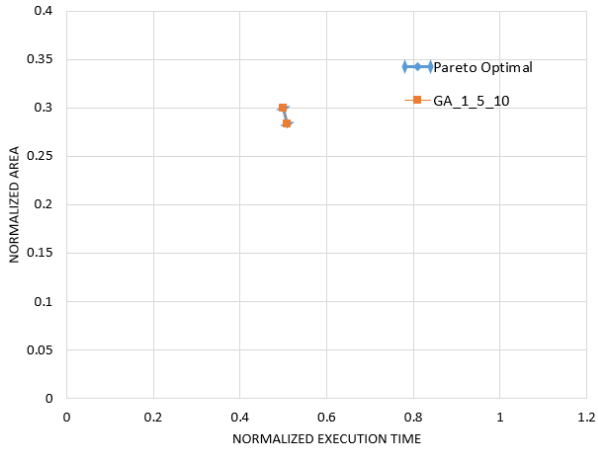
(a) Sobel



(b) FIR



(c) Interpolation



(d) ADPCM

Figure 5.4. System Exploration Trade-off Curves: Pareto optimal Front of Exhaustive DSE vs Pareto Dominant Front of Genetic Algorithm.

Table 5.1. Performance metrics of Fast heuristic Genetic Algorithm with respect to Reference Optimal Solutions

GA Parameters	Performance Metrics	Benchmarks				Average Metrics For GA
		FIR	Interpolation	ADPCM	Sobel	
Mutations=1	**DOM**	0.5	0.5	0.5	0.6	0.52
Parent=5	**ADRS**	0.47	0.33	0.06	0.1	0.24
Count =10	**Spd Up**	4.6	4	1.2	12.6	5.6
Mutations=2	**DOM**	0.75	0.5	0.5	0.6	0.59
Parent=5	**ADRS**	0.48	0.18	0.06	0.16	0.22
Count =10	**Spd Up**	3	2.8	1.17	12	4.74
Mutations=2	**DOM**	0.75	0.75	1	0.6	0.77
Parent=5	**ADRS**	0.48	0.23	0	0.1	0.20
Count =5	**Spd Up**	5.2	4	2	12.5	5.92
Mutations=1	**DOM**	0.75	1	1	0.33	0.77
Parent=3	**ADRS**	0.48	0	0	0.32	0.2
Count =15	**Spd Up**	4.6	4	1.6	21.8	8
Mutations=2	**DOM**	0.5	0.75	1	0.6	0.71
Parent=3	**ADRS**	0.47	0.83	0	0.2	0.37
Count =10	**Spd Up**	4	3.4	1.56	22	7.74

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The thesis first presents OpenCL benchmarks developed to achieve acceleration on SoC-FPGA Architectures and the resulting trends in the acceleration is discussed based on the experiments done on Altera Cyclone V SoC FPGA. The experiments show that the acceleration saturates after a certain input data size due to the communication overhead between the Processor and the FPGA.

Secondly, A fast and heuristic method to explore the design space is developed and its performance is analyzed by comparing with Optimal solutions obtained from Exhaustive DSE. Based on the experiments, an average dominance of 0.7 and an average ADRS of 0.2 at a speed up of 6 times is observed.

6.2 Future Work

The future scope of this work could be experimenting with wider range of benchmarks. The current benchmarks are designed to work only on a single Kernel FPGA device. The benchmarks can be upgraded to experiment on platforms with multiple FPGA devices because most of the current applications (for example in Data Servers) need multiple SoC systems working together.

Other Fast Heuristic methods like In-Situ, Simulation Annealing or Machine Learning algorithms can be used for exploration. There is scope for comparing the efficiency of these algorithms for exploration in the design space.

REFERENCES

[1] CoreTech. Fpga architectures comparision, . URL http://ee.sharif.edu/~asic/Docs/fpga-logic-cells_V4_V5.pdf.

[2] Altera. Cyclone v habndbook, . URL https://www.altera.com/en_US/pdfs/literature/hb/cyclone-v/cv_5v1.pdf.

[3] Khronous. Opencl specification. URL https://www.khronos.org/registry/OpenCL/specs/opencl-2.0.pdf.

[4] Jonathan Tompson and Kristofer Schlachter. An introduction to the opencl programming model. *Person Education*, 49, 2012.

[5] INTEL. Intel fpga opencl sdk programming guide cyclone v. URL https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf.

[6] Intel Altera. Intel opencl optimzation, . URL https://www.altera.com/products/design-software/embedded-software-developers/opencl/developer-zone.html#design-examples.

[7] Wikipedia. Mooore's law, . URL https://en.wikipedia.org/wiki/Moore%27s_law.

[8] Wikipedia. Breakdown of dennard scaling, . URL https://en.wikipedia.org/wiki/Dennard_scaling.

[9] EETimes. An introduction to offloading cpus to fpgas hardware programming for software developers. URL http://www.eetimes.com/document.asp?doc_id=1280560.

[10] Paulo Possa, David Schaillie, and Carlos Valderrama. Fpga-based hardware acceleration: A cpu/accelerator interface exploration. In *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*, pages 374–377. IEEE, 2011.

[11] Intel. Intel acquisitaion of altera. URL https://newsroom.intel.com/press-kits/intel-acquisition-of-altera/.

[12] Microsoft. Microsoft catapult project. URL https://www.microsoft.com/en-us/research/project/project-catapult/.

[13] Xilinx. Xilinx acceleration. URL https://www.xilinx.com/products/design-tools/acceleration-zone.html.

[14] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 365–376. ACM, 2011.

[15] supercomputing conference '15. Is the future cpu+gpu or cpu+fpga. URL https://insidehpc.com/2015/12/does-the-future-lie-with-cpugpu-or-cpufpga/.

[16] Jeff Burt. Intel begins shipping xeon chips with fpga accelerators. URL http://www.eweek.com/servers/intel-begins-shipping-xeon-chips-with-fpga-accelerators.

[17] Wayne Wolf, Ahmed Amine Jerraya, and Grant Martin. Multiprocessor system-on-chip (mpsoc) technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, 2008.

[18] ITRS. The international technology roadmap for semiconductors on soc. URL http://www.itrs2.net/2013-itrs.html.

[19] Robert C Minnick. A survey of microcellular research. *Journal of the ACM (JACM)*, 14(2):203–241, 1967.

[20] Ian Kuon, Russell Tessier, and Jonathan Rose. Fpga architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, 2(2):135–253, 2008.

[21] CoreTech. Fpga architectures overview, . URL https://www.pdx.edu/nanogroup/sites/www.pdx.edu.nanogroup/files/FPGA-architecture.pdf.

[22] Texas Instruments. Multicore socs stay a step ahead of soc fpgas. URL http://www.ti.com/lit/wp/spry296/spry296.pdf.

[23] C-T Hwang, J-H Lee, and Y-C Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(4):464–475, 1991.

[24] Khronous Group. Opencl overview. URL https://www.khronos.org/opencl/.

[25] khronos. Opencl quick reference guide. URL https://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf.

[26] Terasic. Terasic de1soc board. URL http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=836&PartNo=2.

[27] ALTERA. Altera cyclone v. URL https://www.altera.com/products/fpga/cyclone-series/cyclone-v/overview.html.

[28] Intel Altera. Intel fpga opencl sdk best practice guide. URL https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf.

[29] Benjamin Carrion Schafer and Anushree Mahapatra. S2cbench: Synthesizable systemc benchmark suite for high-level synthesis. *IEEE Embedded Systems Letters*, 6(3):53–56, 2014.

[30] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. An approach for effective design space exploration. In *Monterey Workshop*, pages 33–54. Springer, 2010.

[31] Quentin Gautier, Alric Althoff, Pingfan Meng, and Ryan Kastner. Spector: An opencl fpga benchmark suite. In *Field-Programmable Technology (FPT), 2016 International Conference on*, pages 141–148. IEEE, 2016.

[32] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. An efficient design space exploration methodology for multiprocessor soc architectures based on response surface methods. In *Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008. SAMOS 2008. International Conference on*, pages 150–157. IEEE, 2008.

BIOGRAPHICAL SKETCH

Susmitha Gogineni was born in Vijayawada, Andhra Pradesh, India on 22nd August, 1994. She completed her Undergraduate Degree (B.Tech) in Electronics Communication Engineering with distinction from PVP Siddharths Institute of Technology, affiliated by Jawaharalal Nehru Technological University(JNTU), Kakinada. She joined The University of Texas at Dallas to pursue her Masters in Electrical Engineering (MSEE) in August 2015. She joined DARC Lab (Design Automation and Reconfigurable Computing Lab) under Dr. Benjamin Carrion Schafer, in January 2017 to research on Hardware Acceleration on FPGAs. Her research interests include Computer Architecture, Re-configurable Computing, Hardware Acceleration and High Performance Computing (HPC).

CURRICULUM VITAE

Susmitha Gogineni

August, 2017

Contact Information:

Department of Electrical Engineering
The University of Texas at Dallas
800 W. Campbell Rd.
Richardson, TX 75080-3021, U.S.A.

Voice: (469) 763-5200
Email: sxg155930@utdallas.edu

Educational History:

B.Tech., Electronics & Communication Engineering, P.V.P Institute of Technology, Vijayawada, India, 2015
M.S, Electrical Engineering, The University of Texas at Dallas, 2017

Efficient Hardware Acceleration on SoC-FPGA using OpenCL
MS Thesis
Electrical Engineering Department, The University of Texas at Dallas
Advisor: Dr. Benjamin Carrion Schafer

Employment History:

Design Verification Intern, Palo Alto Networks, May 2016 – August 2016