POLITECNICO DI MILANO

Facoltà di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

The exaFPGA base-system

A streaming multi-FPGA system for High Performance Computing

Relatore: Prof. Marco Domenico SANTAMBROGIO

Correlatore: Dott. Ing. Riccardo CATTANEO

Tesi di Laurea di:

Pietro Giuseppe Bressana

Matricola n. 804793

Anno Accademico 2014–2015

This page intentionally left blank

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Application Engine
AEH	Application Engine Hub
AFU	Accelerator Function Unit
ALU	Arithmetic Logic Unit
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BB	Basic Block
BRAM	Block RAM
CAPI	Coherent Accelerator Processor Interface
CAPP	Coherent Accelerator Processor Proxy
CDC	Clock Domain Crossing
CLB	Configurable Logic Block
CN	Cluster Node
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DDR	Double Data Rate
DDR3	Double Data Rate Type Three

DFE DataFlow Engine

DMA Direct Memory Access

DSP Digital Signal Processing

ECC Error-Correcting Code

eDRAM embedded Dynamic Random-Access Memory

FF Flip Flop

FIFO First In First Out

Flex Flexible Subsystem

FLOP Floating Point Operation

FLOPS Floating Point Operations Per Second

FMC FPGA Mezzanine Card

FPGA Field Programmable Gate Array

FSM Finite State Machine

FW Firmware

GC Geometry Core

GCC GNU Compiler Collection

GPGPU General Purpose Graphic Processing Unit

GPU Graphic Processing Unit

GRAPHITE GIMPLE Represented As Polyhedra with Interchangeable Envelopes

HCMI Hybrid-Core Memory Interconnect

HDD Hard Disk Drive

HDL Hardware Description Language

HTIS High-Throughput Interaction Subsystem

HLS High Level Synthesis

HMC Hybrid Memory Cube

HPC High Performance Computing

HW Hardware

ICB Interaction Control Block

IDE Integrated Development Environment

ILP Instruction-level parallelism

IOB Input-Output Block

IP Intellectual Property

IR Intermediate Representation

ISA Instruction Set Architecture

ISL Iterative Stencil Loop

ICT Information and Communications Technology

IVR Integrated Voltage Regulator

JTAG Joint Test Action Group

LUT Look-Up Table

MC Memory Controller

MCU Micro Controller Unit

MIC Many Integrated Core

MIG Memory Interface Generator

MUX Multiplexer

OS Operating System

PC Personal Computer

PCB Printed circuit board

PCIe Peripheral Component Interconnect Express

PL Programmable Logic

PoCC Polyhedral Compiler Collection

PPIM Pairwise Point Interaction Module

PS Processor System

PSL Power Service Layer

RAM Random Access Memory

RAU Remote Access Unit

RTL Register-Transfer Level

RX Receiving

SDRAM Synchronous Dynamic Random-access Memory

SFU Special Functions Unit

SIMD Single Instruction Multiple Data

SLI Scalable Link Interconnect

SM Streaming Multiprocessor

SoC System on a Chip

SP Streaming Processor

SSD Solid State Drive

SST Streaming Stencil Time-step

SW Software

TDP Thermal Design Power

TPC Texture/Processor Cluster

TX Transmitting

USB Universal Serial Bus

VPU Vector Processing Unit

x8 Eight-Lane

Summary

The energy demand of the overall ICT industry amounts to about the 5% of the world's total energy consumption and it is still growing considerably[55]. High Performance Computing (HPC) systems are responsible for a large portion of the energy resources demanded by ICT. Indeed, today's HPC heterogeneous architectures provide a great computing power, at the cost of an enormous energy consumption, which is due both to the Hardware (HW) accelerators *i.e.* Graphic Processing Units (GPUs) and co-processors and to other devices that are not in charge of performing heavy computations, such as the cooling infrastructure, the memory subsystem and the interconnections [43].

Next generation supercomputing platforms will be required to deliver exascale performances, thus increasing the number of performed computations and the overall power consumption. However, the 20MW power budged set by the U.S. Department of Energy to the future HPC platforms [43] in order to limit the total cost of the systems, forces a reduction of at least an order of magnitude to the power consumption of the currently available computing technologies.

The work proposed in this thesis project addresses the aforementioned issues by employing Field Programmable Gate Arrays (FPGAs) as the computing elements for the next generation HPC systems, thus exploiting their intrinsic power efficiency. Moreover, it provides the specifications of a Cluster Node (CN) prototype for the *exaFPGA Infrastructure*, by implementing a high-throughput multi-FPGA pipeline. The performances of the proposed architecture scale linearly with the length of the pipeline and with the number of computing elements implemented on each FPGA.

Moreover, this work enables the use of the Peripheral Component Interconnect Express (PCIe) interface and of the Aurora serial link on the 7-series FPGAs provided by Xilinx, thus simplifying the development of a working multi-FPGA HPC system.

A first experimental version of the proposed architecture has been implemented with two Xilinx *VC707* boards and five benchmarks have been run by employing various types of Streaming Stencil Time-steps (SSTs). Results show how the high scalability of the architecture is guaranteed by the pseudo-linear increase of the throughput and of the power efficiency. Moreover, the low resource utilization allows to reserve a considerable amount of area for the computing logic, thus implementing long SSTs queues, that ensure a pseudo-linear increase in throughput while remaining with constant bandwidth.

Sommario

La richiesta di energia da parte del settore informatico ammonta complessivamente al 5% di tutta l'energia consumata nel mondo ed è in continua crescita[55]. I sistemi High Performance Computing (HPC) sono i responsabili del consumo di una buona parte delle risorse energetiche utilizzate dal settore informatico. Infatti, le odierne architetture per il supercalcolo forniscono una grande potenza di calcolo, al costo di un enorme dispendio di energia, dovuto sia agli acceleratori Hardware (HW) inclusi nei moderni sistemi HPC, ovvero Graphic Processing Unit (GPU) e coprocessori, sia a altri dispositivi che non eseguono computazioni intensive, come l'impianto di raffreddamento, il sistema di memoria e le interconnessioni [43].

La prossima generazione di piattaforme per il supercalcolo sarà chiamata a fornire prestazioni dell'ordine del miliardo di miliardi di operazioni floating point eseguite in un secondo. Si assisterà quindi ad un aumento del numero di calcoli eseguiti e della potenza utilizzata. Tuttavia, il limite di 20MW imposto dal Dipartimento dell'Energia degli Stati Uniti alla potenza consumata dai sistemi di supercalcolo del prossimo futuro [43], richiede una riduzione di almeno un ordine di grandezza della potenza consumata dalle tecnologie attuali.

Il lavoro proposto in questo progetto di tesi affronta i problemi appena citati impiegando le Field Programmable Gate Array (FPGA) come elementi computazionali per i futuri sistemi di supercalcolo, avvalendosi della loro intrinseca efficienza energetica. Inoltre fornisce la specifica di un Cluster Node (CN) per l'infrastruttura del progetto *exaFPGA*, realizzando una pipeline multi-FPGA ad alte prestazioni. Le prestazioni dell'architettura proposta scalano linearmen-

te con la lunghezza della pipeline e con il numero di elementi computazionali implementati su ogni FPGA.

Inoltre, questo lavoro permette l'uso dell'interfaccia Peripheral Component Interconnect Express (PCIe) e della connessione seriale con Aurora sugli FPGA della serie 7 prodotti da Xilinx, semplificando notevolmente lo sviluppo di un sistema HPC multi-FPGA.

Una prima versione sperimentale dell'architettura proposta è stata realizzata con due schede Xilinx VC707 e sono state eseguite cinque sessioni di test utilizzando diversi tipi di Streaming Stencil Time-step (SST). I risultati ottenuti mostrano come la notevole scalabilità dell'architettura sia garantita dall'aumento pseudo-lineare del throughput e dell'efficienza energetica. Inoltre, il ridotto utilizzo delle risorse permette di dedicare una considerevole quantità di area agli elementi computazionali, consentendo quindi di realizzare lunghe catene di SST, che comportano un aumento del throughput, pur mantenendo la banda costante.

1

Introduction

This chapter introduces the context in which the work done in this thesis project has been carried on. Section 1.1 describes this context, namely the High Performance Computing (HPC) field and its applications. The main challenges that the future architectures will be called upon to address while moving towards the exascale era are analysed in section 1.2. Section 1.3 describes the most promising technological advancements that allow to address the presented challenges. Then, section 1.4 presents an overview of the most common heterogeneous technologies, as they will be crucial for the development of the future exascale systems, while section 1.5 focuses on the use of FPGA-based devices in the HPC field, since this seems to be a promising approach in order to meet the presented challenges. Section 1.6 presents the iterative stencil loops, that are commonly found in HPC applications, and the polyhedral model, which will be exploited in the future supercomputing systems, since it enables a number of code transformations and optimizations. Finally, section 1.7 provides the contributions made by the proposed work and an overview of the remaining of this thesis.

1.1 Context

HPC is the branch of computer science which develops new hardware and software technologies in order to achieve substantially higher computing performance with respect to the standard general-purpose computers. In modern

societies HPC is regarded as a key asset for a variety of commercial and research activities and a enabling technology for new discoveries[44]. Scientific research employs HPC to perform simulations of complex molecular systems that are useful to develop new chemical structures and new materials [51]; physicists need to solve compute-intensive problems to discover the behaviour of the primary components of the matter and to improve their knowledge of the universe [54]. Moreover, they can develop innovative energy plants, that exploit renewable energy sources to provide a better efficiency and a reduced pollution; HPC is used for DNA sequencing and for DNA micro array analyses, thus allowing to find the cause of several diseases and to produce new drugs for them [61]. Moreover, the execution of some clinical tests can be accelerated and improved, hence providing detailed results in a shorter time. HPC is currently used also to simulate the human brain at the neural level [8], thus allowing to understand its behaviour; fast climate changes can be simulated with supercomputers to forecast short term events and long term trends; the major oil and gas companies carry out complex computations to detect the presence of hydrocarbons reserves within the layers of the earth's crust [49]; several financial institutes employ HPC systems to perform financial calculations on huge data sets, in order to foresee the trends of the markets [67]; many research centres and companies all over the world perform complex simulations and analyses on their prototypes using supercomputers instead of building expensive samples of their products; finally, governments employ HPC in defence, communications, aerospace and other strategic fields [45]. Thanks to its unmatchable computing capability, HPC technology not only reduces the time taken by complex computations, but also allows to develop new algorithms and to meet a number of previously unfeasible challenges.

Future HPC systems will be required to deliver exascale performance, thus providing more than 10^{18} Floating Point Operations Per Second (FLOPS). Therefore a great performance improvement is needed, since today's best HPC systems[28] can achieve only peta-scale performance, thus providing more than 10^{15} FLOPS. In order to achieve this performance improvement, a number of technological challenges arisen within the HPC field must be addressed properly.

1.2 The Challenges of High Performance Computing

This section describes the main challenges that must be addressed in order to achieve the performance improvement required by the future exascale HPC systems. They deal with power consumption, scalability, memory performance and programming paradigm.

1.2.1 The Power Challenge

About the 5% of the total energy consumed in the world is spent by the ICT sector [55]. Data centres account for a third of the ICT total and this value will increase in the next years. Indeed, current HPC systems can achieve petascale performance at the cost of about ten Mega-Watts of power consumption, thus providing a very low power efficiency. Only a fraction of the consumed power is required by computation, a large amount of power is spent to move the data throughout the infrastructure and for the cooling systems [43].

The power required by the interconnection infrastructure increases with the size of the infrastructure itself, because it has a quadratic dependency to the length of the links. Moreover, by increasing the rate at which the data is transmitted, the power consumption increases as well.

A high power consumption involves both financial and technical problems. Indeed, the cost of the energy has become the main expense when considering HPC systems, as it has overcome the cost of the system itself. Moreover, a enormous quantity of energy flowing through the infrastructure requires a powerful cooling system, which in turn needs a lot of energy to be operated, thus increasing the operational cost of the overall system. For these reasons the U.S. Department of Energy has set a 20MW limit to the power consumption of the future exascale supercomputing systems [43].

A more efficient HPC technology would allow to decrease the overall costs by reducing the amount of energy required to perform computations and by requiring a smaller cooling system, since a reduced amount of heat would need to be dissipated. Power efficiency is the main concern when considering HPC tech-

nology, since it hampers the development of more powerful systems based on the currently available architectures.

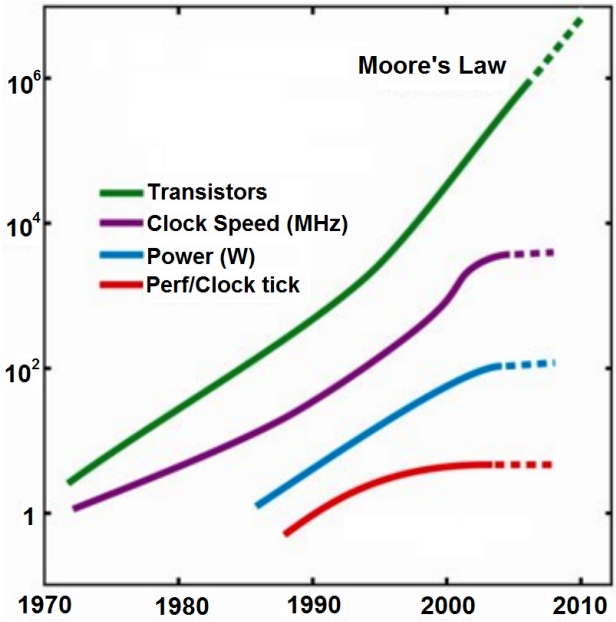


Figure 1.1: Processor power, density and performance trends over the last forty years.

1.2.2 The Scalability Challenge

Scalability is an important aspect of the HPC technology, since it enables the performance increase of the systems. In the past years supercomputers have scaled their performance by improving the clock frequency and the Instruction-level parallelism (ILP) of each of their computing nodes, following the Moore's law, which states that the number of transistors on a chip doubles every 18/24 months. However, this growth in transistor density no longer corresponds to the doubling of the overall performance, due to power consumption and memory performance, and the growth in computing performance can no longer rely on the increase of the single-compute node performance, see figure 1.1.

The maximum power that can be supplied to a single computing device is limited by the maximum heat that the device can dissipate, this limit has already been reached and the clock frequencies of the computing devices has remained flat at some Gigahertz. This power limitation causes also the phenomenon known as *dark silicon* [63]: transistor scaling and voltage scaling are no longer in line

with each other, hence the amount of power that can be supplied to a chip is not enough to operate all the computing units at the same time. Therefore, only a fraction of the entire die can be powered up at a time, causing large idle or heavily underclocked portions of silicon area, hence the term dark silicon. Moreover, computing performance can no longer be increased by adding more nodes to the system, because a bigger system would require an enormous amount of energy. Therefore HPC systems can not be scaled both by increasing the single device performance and by adding more devices to their infrastructure.

1.2.3 The Memory Challenge

Another important challenge is brought by the performance gap between the memory and the computing logic. Indeed, in the past years, memory technology has been optimized to offer the maximum possible density, in order to maximize the amount of data available for the computing devices. However, the high latency provided by the memory has decreased slowly with respect to the computing logic cycle time, therefore the gap between the number of instructions that a processors can execute and the number of memory transfers that can be performed in the same amount of time forces the computing devices to waste time and resources while waiting for data stored in memory.

Moreover, while the number of processing units included into each device increases, the amount of memory ratio with respect to the available computational capacity decreases. This trend is caused by the higher cost of the memory with respect to the computing resources. Indeed, memory density has not increased at the same rate of the computing logic, which in turn has followed the Moore's law. Therefore HPC systems are equipped only with the essential amount of memory, hence restricting the overall performance improvements.

1.2.4 The Programmability Challenge

Programming a today's HPC system is a hard task, as it becomes more and more difficult to efficiently run a sequential code on a thousands node machine. Indeed, in order to extract the adequate amount of parallelism from a code, it has

to be changed or completely rewritten.

Usually, today's developers and HPC users are not trained on parallel programming, therefore they are not able to exploit all the computing power offered by the machines. Moreover, high level programming languages, which fit well for sequential programming, are not suited for the supercomputing field, as they do not allow to write parallel code natively. Finally, some classes of problems require too much effort to be accelerated within the currently adopted paradigm.

1.3 Innovations in High Performance Computing

In the last years a number of new techniques in the supercomputing field have arisen. They span all the aspects of the HPC technology, such as memories, interconnections, cooling, programming paradigms and computing architectures, in order to address the challenges provided by today's technologies.

The designers of HPC systems try to hide the performance gap between the memory and the computing logic by inserting a increasingly longer memory hierarchy composed of different cache levels. Moreover, one of the solutions adopted to reduce the power consumption of the communication infrastructure consists of replacing the entire infrastructure with fiber-optic links, that provide high speed and low power consumption. When considering the heat dissipation problem, innovative cooling systems that exploit renewable energy instead of the traditional power sources have been constructed. Moreover, new data centres arise within suitable environments providing natural cooling capabilities, such as cool water availability or cold weather. One of the most important trends, which is presented in sections 1.4 and 1.5, concerns computing architectures, as it suggests to build heterogeneous systems by exploiting different kinds of processing elements instead of the general purpose processors only. Although this solution will require a huge effort in order to be fully implemented, it seems to be promising for the exascale approach, with respect to the power efficiency and to the scalability. On the software side, new parallel programming languages are under development and new approaches are being tested in order to optimize the still

unmanageable classes of problems. In this field a great research effort has been spent on the iterative stencil loops, since they find application in many HPC algorithms, and on the polyhedral model, which enables a number of optimizations, that can be easily applied also to ISL-based codes. They are both presented in section 1.6.

1.4 Heterogeneous Systems

As of June 2015, the two best HPC systems [28] where based on heterogeneous architectures. In the near future, many other supercomputers will be built by employing different kinds of accelerators. This approach offers to the developers the more appropriate processing elements for each of the computational tasks in which an application can be divided. Indeed, each computing device composing a heterogeneous system features an optimized architecture, which allows to execute a suitable task in the most efficient way. Moreover, HPC developers can exploit a greater flexibility, both on the software side and on the hardware side.

Next heterogeneous HPC systems will provide a higher computing performance with a lower power consumption with respect to the currently available homogeneous machines. The introduction of heterogeneous computing architectures is due to the inability to solve the aforementioned issues simply by improving the already available homogeneous supercomputers. Indeed, homogeneous architectures are not able to satisfy all the different computational requirements of an application with the same level of efficiency. Therefore, today's HPC systems generally achieve only a fraction of their peak performance during their utilization. The maximum performance provided by the processors can be reached only when particular suitable tasks are executed.

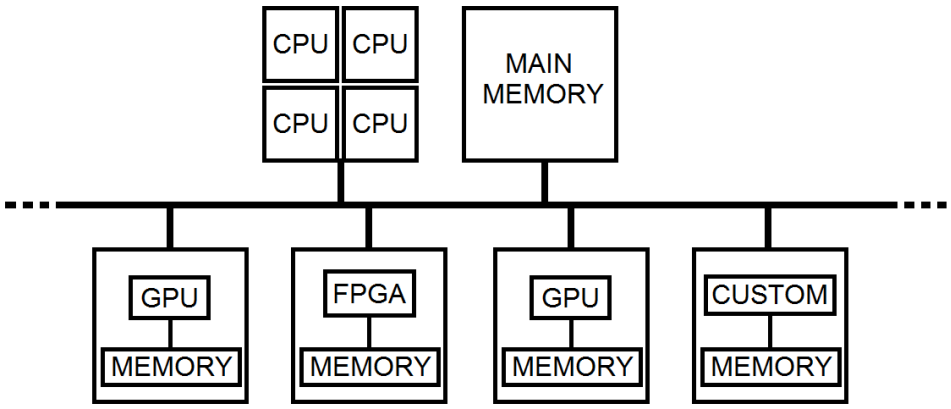


Figure 1.2: Block diagram of an heterogeneous architecture.

The architecture of a heterogeneous computing system, shown in figure 1.2, consists of a number of different processing elements connected to a computing machine, which in turn is composed of one or more general purpose processors and a main memory unit. The accelerators, the processors and the memory are connected through a high-performance link. The whole system can fit into a standard cluster node unit and hundreds or thousands units can be interconnected to build a complex supercomputer.

In order to better exploit all the computing power offered by the system, the movement of the data and the execution of general purpose tasks, such as the Operating System (OS), is performed by the Central Processing Units (CPUs), since they can manage complex control flows more efficiently than the other computing architectures. On the contrary, the most compute-intensive tasks are processed by the dedicated accelerators.

The accelerator units can be of different nature, depending on the characteristics of the application that has to be processed. The heterogeneous components used to build today's HPC accelerators are General Purpose Graphic Processing Units (GPGPUs), Field Programmable Gate Arrays (FPGAs) and Custom Processors.

GPGPUs are high performance parallel computing devices based on Graphic Processing Units (GPUs), that feature an optimized Single Instruction Multiple Data (SIMD) architecture. They can achieve a high floating-point performance, a

high memory bandwidth and a high power efficiency, therefore they outperform general purpose units when processing huge sets of of floating-point data with modest need of control flow and data caching.

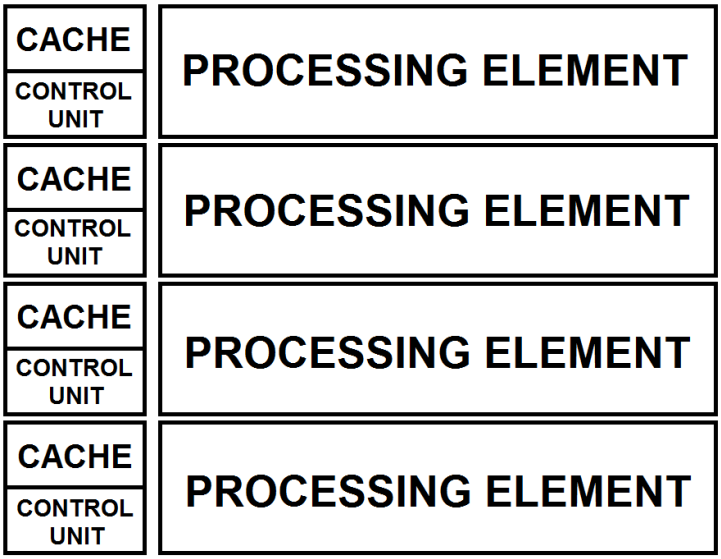


Figure 1.3: Block diagram of a GPGPU.

The block diagram shown in figure 1.3 illustrates the main components of a GPGPU device. The architecture includes a number of identical units, each one featuring a processing element, a cache memory and a control unit.

Each processing element comprises a pipeline of dedicated processing units that allows to compute basic floating-point arithmetic operations with a very high throughput. The cache memory provides a intermediate high-speed memory level between the external Random Access Memory (RAM) and the processing element. The control unit manages the movement of the data throughout the architecture and, in modern GPGPUs, it can also schedule the execution of the threads, thus enabling a better exploitation of the processing units.

These elements are replicated several times into the architecture in order to perform simultaneous computations on large amounts of data. GPGPU devices are usually coupled with a off-chip dedicated high performance Double Data Rate (DDR) memory, that allows to temporarily store the data to be processed.

Today's GPGPU devices offer new hardware features that make them more suitable for HPC, for instance:

- IEEE-compliant Fast Double-Precision and Single-Precision Floating-Point Arithmetic, providing standard floating-point arithmetic operations at roughly the double of the speed with respect to the general purpose processors;
- Memory Hierarchy, consisting of a L1 cache coupled to each processing element, a common L2 cache and a off-chip large amount of dedicated DDR memory;
- 64-Bit Addressing and Unified Address Space, making easier for the developers to use the different types of memory included into the system and to manage the pointers for input and output data;
- Error-Correcting Codes (ECCs), providing a more robust platform, in particular for critical HPC applications;
- Fast Context Switching, which allows to manage different contexts and to quickly switch from one to another using dedicated hardware units.

GPGPU accelerators can be programmed using a number of standard languages that provide special parallel libraries. By exploiting this feature, HPC developers can accelerate the most compute-intensive tasks of an application directly by modifying the source code of the application itself in order to exploit the dedicated parallel functions provided by the GPGPU units.

In the last years GPGPUs have been greatly improved, thanks to the available GPU technology and to the significant need for programmable high performance accelerators. The most innovative devices, designed by Nvidia and AMD, belong to the Tesla family and to the FirePro series respectively. In particular, Nvidia provides a broad range of accelerators based on the Testla[53], Fermi[18] and Kepler [19] microarchitecture, that can be programmed with a dedicated language named CUDA[17]. The solutions provided by Nvidia are described in chapter 2.

Although GPGPUs offer high parallel computing capabilities and a good power efficiency, they are based on a fixed architecture. Therefore if an algorithm is unable to fully exploit the GPGPU architecture, the efficiency of the system is greatly reduced, due to the bad utilization of its resources.

FPGAs are integrated circuits that can be electrically programmed after manufacturing, by providing a software specification of the digital circuit that they have to implement. FPGAs can be configured several times, thus allowing to change their functionality. Some devices can be reconfigured while operating, by programming a portion of their resources in order to implement a different function.

These devices represent a cheaper solution with respect to Application-Specific Integrated Circuits (ASICs), since they cost around a few tens to a few thousand dollars and they take less than a second to be configured, while ASICs require a long development time and a great amount of money to be produced. However, FPGAs become a convenient option when medium volumes of devices have to be produced. Moreover, the great flexibility of the FPGAs makes them slower and more power consuming than their ASIC counterparts.

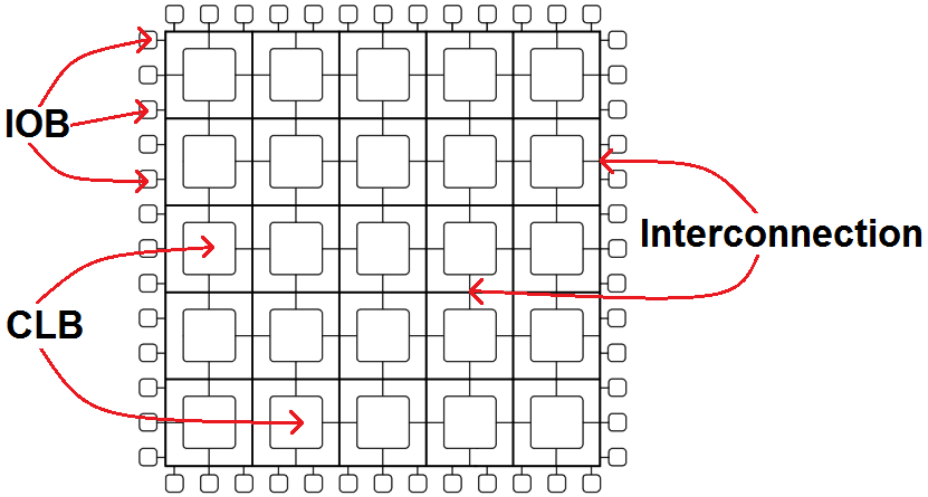


Figure 1.4: Block diagram of a FPGA.

A standard FPGA device, depicted in figure 1.4, is composed of an array of Configurable Logic Blocks (CLBs) linked by a programmable interconnection, which occupies most of the area, and a number of Input-Output Blocks (IOBs), that provide the off-chip connections.

Each CLB, which features a variety of input and output ports, provides a customizable logic function and can be connected with other CLBs in order to

implement a full computing unit. The architecture of a CLB, shown in figure1.5, is composed of a Look-Up Table (LUT) connected to a Flip Flop (FF) and to a Multiplexer (MUX). A LUT can be configured and reconfigured in order to implement on the output pin any function of its input pins. The MUX allows to select the data to be forwarded to the output pin of the CLB unit, since it receives at its input pins both the data computed by the LUT and the data stored into the FF. The flexibility offered by the LUTs provides a good compromise between a fine-grained and a coarse-grained architecture. Indeed, each LUT-based CLB can implement a logic function, therefore more complex functions can be obtained by connecting multiple CLBs and simpler functions can be provided without wasting too many resources.

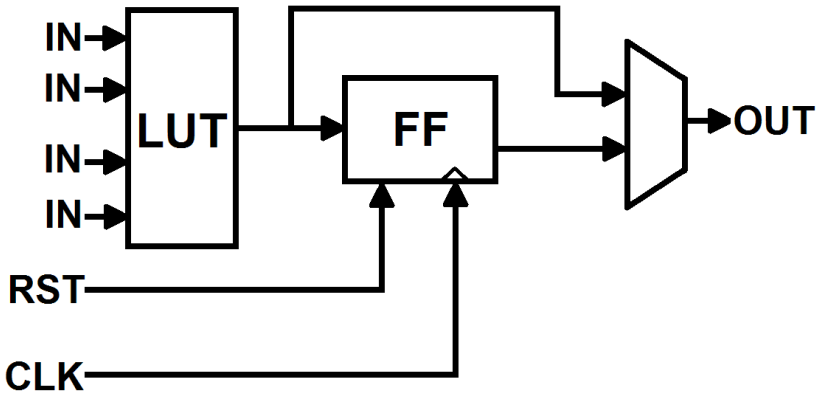


Figure 1.5: A Configurable Logic Block.

The IOBs are connected to all the input/output pins of a FPGA chip. They allow to interface the on-chip digital circuit with many other off-chip devices and complete computing systems. Moreover, they provide the external clock and reset signals, and the power source to the device. Finally, some dedicated IOBs can be used to program the FPGA and to perform debug functions.

CLBs and IOBs are connected to each other through the programmable routing architecture, which is composed of a number of long and short wires and programmable switches. The network presents a hierarchical and flexible structure, that allows to implement different digital circuits, keeping the design performance as high as possible. Moreover, a variety of dedicated paths uniformly

distributes the clock and reset signals through the entire area of the FPGA, thus maximizing the synchronization between the signals. Today's FPGAs combine IOBs with high performance transceivers that provide high throughput interconnections for special applications, such as Peripheral Component Interconnect Express (PCIe) interfacing, serial board-board transmission and Ethernet communication.

FPGAs feature also a number of ASIC units, implemented on the same silicon die on which the basic flexible components are placed. These hardware units provide standard high performance functions, such as digital signal processing and floating-point computing and implement common interconnection protocols like PCIe. ASIC technology is also used to include high speed RAM units on the device.

Typically, FPGA-based HPC accelerators feature a large amount of DDR memory, which is used to store the data to be processed by the FPGA, a flash memory, that stores the configuration file for the device and a variety of high speed interfaces linked to the on-chip transceivers, such as PCIe, Ethernet and serial links. This configuration allows to exploits the I/O bandwidth that the FPGA device can achieve, also by employing more I/O interfaces as a single high performance unit.

Recently, FPGA-based accelerators have become of great interest for the HPC developers, since they offer a great flexibility both on the hardware side and on the interconnection side. Thanks to their flexibility, FPGAs can be much more power efficient with respect to GPUs, in particular when special functions have to be accelerated and when the executed function is not able to achieve the full utilization of the hardware accelerator. Indeed, the most of the power consumption of an FPGA is due to the digital circuit implemented on the device. The unused logic receives no clock signal, therefore it consumes very little power, thus increasing the overall power efficiency.

Although the architecture of an FPGA is flexible, it is also complex to manage and to program. Indeed, as opposed to GPUs, FPGAs need dedicated low-level Hardware Description Languages (HDLs) to be programmed, that are unfamiliar

to the most of the software developers, since they are trained with common high-level programming languages. Therefore, to be efficiently employed, FPGAs require both hardware and software skills and the ability to manage them simultaneously, the so called *hardware/software co-design*.

In order to make the FPGA development easier, recent devices can be programmed also using high-level languages, such as C and C++, by exploiting High Level Synthesis (HLS) tools. Moreover, some devices support OpenCL specifications, by programming a suitable hardware architecture on the FPGA, that includes also the hardware implementation of the software kernels provided by the developers.

Today, a number of companies provide FPGA-based HPC accelerators and supercomputers. The devices developed by two of the main manufacturers, Pico Computing[21] and Convey Computer[4], are described in chapter 2.

Custom processors are ASIC units designed to process fixed computational workloads. They can be used in the HPC field to speed-up compute-intensive tasks, by providing higher performance and power efficiency with respect to GPGPUs and FPGAs, since they are optimized to implement only the function that they have to execute. However, due to the employed ASIC technology, custom processors are very expensive and are not flexible at all. Therefore, they are used only to accelerate extremely critical applications, that require custom highly-optimized supercomputers instead of huge GPGPU-based or FPGA-based standard HPC systems.

The machines based on custom processors are usually composed of a single CPU-based unit that manages the computation running on a number of cluster nodes, each one including a few tens of custom processors. Some devices feature a limited amount of on-chip programmable SIMD units and simple general purpose processors, that provide a little of flexibility to the architecture, by allowing the system to adapt to the changes made to the algorithms.

Custom accelerators require a great effort in order to be programmed, because they are composed of non-standard processing units that need a variety of programming languages and techniques to be operated.

Chapter 2 illustrates in detail the architecture of the Anton[51][52] supercomputer, a custom HPC system optimized for high-performance molecular dynamics simulation.

Next section focuses on the use of FPGA-based accelerators in the HPC field, that can be connected together in order to build multi-FPGA supercomputers, by exploiting the high power efficiency, scalability and flexibility provided by the FPGA technology.

1.5 FPGA-based Systems

In past years, FPGAs have achieved a very little utilization in the HPC field, mainly due to the programming effort that they required to be efficiently employed. Recently, the manufacturers of programmable logic devices have facilitated the development of complex FPGA-based systems, by raising the programming level from basic digital components to entire processing units and by providing standard interfaces, that allow to connect the FPGAs with a broad range of off-the shelf devices. Therefore, today's FPGAs can be programmed with usual programming languages and, thanks to their inherent scalability, they can be connected together to implement complex multi-FPGA systems.

FPGA-based HPC architectures can be quickly changed in order to implement completely different functions, by exploiting the great flexibility offered by the programmable logic. Moreover, they can achieve a high power efficiency, which will be a significant capability of the future supercomputing systems. The high-bandwidth provided by the FPGAs is a crucial feature, which is essential when processing enormous amounts of data and when moving the data through complex interconnections. Finally, dynamic reconfiguration is a unique asset of the FPGA technology, which allows to implement a variety of functions on the same computing device, by dynamically switching from one to another while the system is running.

These advancements have convinced many manufacturers to develop new supercomputing solutions based on FPGAs. Currently, the range of available

FPGA-based HPC products includes accelerators, chassis and complete supercomputing platforms.

FPGA-based accelerators are usually implemented as standard Personal Computer (PC) cards, that provide one or more high-end FPGAs coupled with a PCIe interface, a on-board DDR memory, one or more network connectors, such as Ethernet, and some other proprietary links. These devices can be customized by adding more on-board memory or, in some cases, by changing the employed FPGA chip.

Due to their features, the accelerators can be used both in workstations, to accelerate limited computations, and in supercomputing systems, to boost the overall performance. Moreover, thanks to the available interfaces, FPGA accelerators can be connected together to build complex HPC platforms.

Chassis are complete HPC machines featuring one or more server-class general purpose processors strictly coupled with a number of FPGA accelerators. Although they are usually based on standard rack units and PCIe accelerator boards, in some cases they include custom boards that carry both the processors and the FPGAs. Chassis can be used both as stand-alone supercomputers and as high-performance cluster nodes in complex HPC systems.

A FPGA-based HPC platform offers a complete environment that provides both a supercomputing infrastructure and a dedicated programming framework. It also includes additional tools, that allow to manage the system and to detect and to fix hardware and software faults. Currently, FPGA-based platforms are not widely used in the supercomputing field, since they are usually developed by academic research centres or by big companies that employ them for their own needs, without making them commercially available.

The best known exception is Maxeler [14], which offers a commercial dataflow FPGA-based platform for financial, oil and gas and scientific computing.

Chapter 2 also describes in detail NetFPGA [16], a academic research platform for rapid prototyping of computer network devices, based on single-board FPGA accelerators and Catapult [15], a server-based HPC architecture developed by Microsoft in order to accelerate Bing's search engine, by coupling standard

cluster nodes with custom FPGA accelerators.

1.6 Iterative Stencil Loops

Iterative Stencil Loops are iterative kernels that repeatedly perform a sequence of sweeps, called time-steps, through a given array, whose elements are updated in each time-step using neighbouring elements in a fixed pattern, called stencil, see figure 1.6. The shape of the neighbourhood used during the updates depends on the application itself.

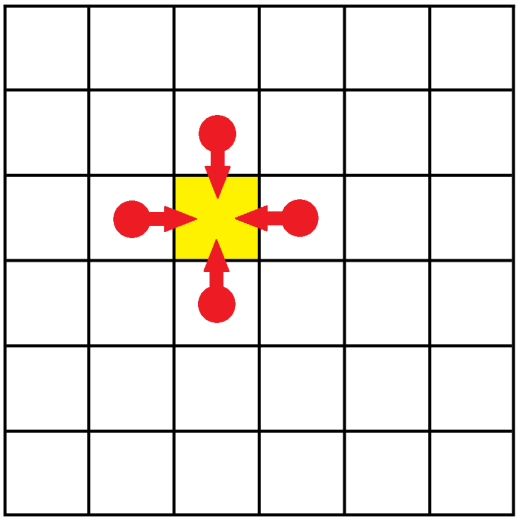


Figure 1.6: Four-neighbour stencil in a 2D array.

Iterative Stencil Loops (ISLs) belong to the class of the static affine codes, that can be optimized using the polyhedral model and are commonly found in scientific and engineering HPC applications, such as particle interaction simulation and differential equations solving. If an ISL has no true data dependencies between the points of the array in the same time-step, then every element can be independently computed from each other. Therefore the updating of the elements is easily parallelizable, thus allowing a variety of optimizations. After being optimized, the code can be implemented in hardware by exploiting HLS tools, that allow to generate a hardware implementation of a C code without writing HDL code manually.

Polyhedral Model

The polyhedral model [47], also known as polytope model, is a mathematical framework that provides automatic loop nest restructuring, optimization and parallelization by treating each loop iteration within nested loops as lattice points inside mathematical objects called polytopes. It then performs affine transformations or more general non-affine transformations and converts the transformed polytopes into equivalent, but optimized, loop nests.

Under certain regularity conditions, the performance of the optimized code can be greatly enhanced in terms of throughput, execution time, communication channels and memory requirements.

The polyhedral model is based on algebraic representation of programs, therefore it allows to execute complex sequences of optimizations by performing a number of polyhedral transformations, that modify the original order of the operations. An overview of a few loop transformations is provided.

- Loop Reversal: it reverses the direction in which the loop traverses its iteration range, thus allowing the execution of further optimizations;
- Loop Interchange: it exchanges the position of two loops in a loop nest. It is used to improve the accesses to arrays and to interchange parallel loops with non parallel ones;
- Loop Fusion: it combines two loops body, in order to enhance data reuse, to reduce loop overhead and to remove synchronization between parallel loops;
- Loop Distribution: it is the inverse of the Loop Fusion transformation. Indeed, it breaks a single loop into multiple loops, iterating over the same index range. This transformation can be used to enable further transformations and to allow partial parallelization.

Polyhedral transformations, including the aforementioned ones, are provided by a wide variety of tools, that follow different approaches. Although they pro-

vide source code optimizations, some of them implement also a complete compiler.

Polyhedral Compiler Collection (PoCC)[24] is a flexible and model-driven compiler, that embeds a number of tools for polyhedral compilation. It can generate an optimized C code starting from a C source code and it can compile the optimized code in order to generate an executable binary.

PoCC comprises a variety of free software dedicated to polyhedral compilation, including:

- CLAN [3], a software tool that generates a polyhedral Intermediate Representation (IR) of a program, starting from its source code, which is written using one of the common high level programming languages, such as C, C++, C# or Java. The IR can be manipulated by other tools in order to perform complex analyses, restructurations and optimizations;
- CANDL [24], a software tool that computes the set of statement instances in dependence relation, starting from the polyhedral representation of a static control part of a program. The output generated by CANDL can be provided to other tools in order to build program transformations respecting the original program semantics;
- PLUTO [23], an automatic parallelizer and locality optimizer for affine loop nests, that performs source to source transformations on C programs, by improving coarse-grained parallelism and data locality simultaneously.

GIMPLE Represented As Polyhedra with Interchangeable Envelopes (GRAPHITE) [59] is a framework for high-level memory optimizations using the polyhedral model. Coupled with GNU Compiler Collection (GCC), it implements a general-purpose compiler to build on full-scale polyhedral compilation techniques. Its main goal is to bring more high-level loop optimizations to GCC, based on polyhedral representations of loop nests.

1.7 Thesis Contributions and Outline

Within the HPC field, this thesis project focuses on the acceleration of ISL-based computations using FPGA technology. In particular, it aims to develop a multi-FPGA architecture, which is able to exploit the Streaming Stencil Time-step (SST) units [7] designed as part of the exaFPGA project [6] in order to provide a high-throughput programmable HPC system, that features a high power efficiency and an explicit scalability mechanism.

The proposed work addresses the aforementioned problem by providing a dedicated multi-FPGA architecture based on the Xilinx VC707 [29] evaluation board. The system has been designed, constructed and tested using a variety of SSTs, that implement common HPC algorithms in hardware. It consists of a chain of boards hosted by a cluster node, that allows to move the data through the computing cores implemented on the FPGA boards. The implemented architecture allows to use the SSTs in the form of hardware Intellectual Property (IP) cores that can be queued and added to the multi-FPGA infrastructure.

The power efficiency of the system is ensured by design, as it is based on FPGA technology, which requires very low power to perform computations. The use of FPGA-based accelerators delivers also good flexibility to the system and allows to implement different algorithms without changing the multi-FPGA infrastructure. Moreover, the system can be easily scaled both by adding more SSTs to each board and by adding more boards to the chain.

A quantitative model has been designed in order to analyse the power efficiency trend of a multi-FPGA system. This model has been validated against the experimental results.

The computation and the movement of the data through the infrastructure are completely managed by a custom application running on the host system. Users only need to modify the input data for the application, simply by changing few lines in the source code, which is written using C language. Therefore, no further firmware, driver or custom software has to be developed.

The remaining of this thesis is organized as follows. Chapter 2 offers a detailed description of the state of the art technologies employed in the HPC field.

The background knowledge provided by this chapter is necessary to comprehend the work proposed in this thesis project. In chapter 3 a detailed description of the proposed architecture is presented. The evaluation of the proposed design is provided in chapter 4, alongside the description of a quantitative model for the power efficiency and its validation against the experimental results. Finally, chapter 5 provides the conclusions of this thesis project and some considerations on possible future works.

2

State of the Art

This chapter offers a detailed description of the state of the art technologies employed in the High Performance Computing (HPC) field. The background knowledge provided by this chapter is necessary to comprehend the challenges mentioned in chapter 1, the current trends in HPC and the already developed solutions. Moreover, it allows to understand how the proposed work takes part to the journey towards exascale computing.

In particular, section 2.1 presents the most advanced heterogeneous computing architectures, dealing with Graphic Processing Units (GPUs), Central Processing Units (CPUs), Field Programmable Gate Arrays (FPGAs) and custom processors. In section 2.2 multi-FPGA architectures are treated, as they allow to perform complex computations over huge sets of data. This thesis project proposes a multi-FPGAs architecture as well. Finally, Maxeler Architecture is described in detail in section 2.3, as a widely-used HPC platform, that implements a multi-FPGA streaming system, like the design proposed in this thesis project.

2.1 Heterogeneous Systems

This section provides a detailed overview of the state of the art heterogeneous HPC systems, grouped based on the component that best characterizes their architecture. Some of the systems could be included under different categories at the same time, due to their strong heterogeneous nature, in that case the pro-

posed arrangement is purely subjective.

2.1.1 GPU-based Systems

GPU-based systems focus on the key role of the GPU devices, used as massively parallel accelerators. The general purpose CPU is not directly involved in the computation, since it only takes care of the movement of the data.

NVIDIA GPGPU

In the GPGPU computing field, the state of the art coincides with the solutions provided by NVIDIA. Indeed, NVIDIA Tesla has been the first GPU developed both for graphics, both for HPC, as it has been paired with Compute Unified Device Architecture (CUDA). Tesla has been followed by Fermi and by Kepler, that has been adopted by many Top500 [28] HPC systems. As of June 2015, the second most powerful supercomputer exploits Kepler GPGPU architecture. After Kepler, NVIDIA has released Maxwell GPU architecture, which is designed primarily for fast graphics performance and single precision consumer compute tasks, therefore it is not suited for HPC.

Tesla [53], released by NVIDIA in 2006, has been the first high-level programmable GPU architecture that can perform general purpose calculations alongside graphic computations. Moreover, it has been the first unified shading architecture developed by NVIDIA. Indeed, all the computational units have been designed to handle any type of shading tasks, thus enabling dynamic load balancing of the workload on the available computing units. This generality has opened the door to a completely new GPU parallel-computing capability, leveraged by the introduction of CUDA parallel programming language.

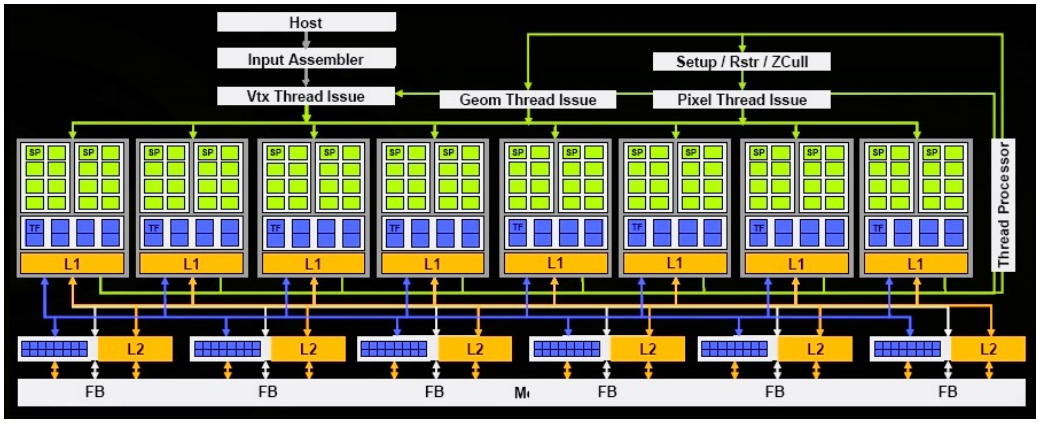


Figure 2.1: Tesla microarchitecture.

Figure 2.1 shows the structure of Tesla microarchitecture.

The work flows from the top to bottom, starting at the host interface with the system PCIe bus, traversing the computing units and reaching the bus that connects the GPU to the external memory. The main unit of this architecture is the Texture/Processor Cluster (TPC), that contains a number of computing elements, an interconnection network, a cache memory and a shared memory. Each TPC includes two Streaming Multiprocessors (SMs), unified graphics and computing multiprocessors that execute graphics shader thread programs and GPU computing programs.

The unified SMs concurrently execute different thread programs. Their multithreaded architecture can create, manage, schedule and execute threads in groups of 32 threads called warps. Each SM manages a pool of 24 warps, with a total of 768 independent threads. SM architecture is similar to Single Instruction Multiple Data (SIMD) design, which applies one instruction to multiple data lines. The difference is that this architecture applies one instruction to multiple independent threads in parallel, not just multiple data lines. A thread in a warp can take conditional branches, when all paths complete, the threads reconverge to the original execution path.

Each SM contains eight Streaming Processors (SPs) cores, two Special Functions Units (SFUs), a multithreaded instruction fetch and issue unit, a cache, a shared memory and a low-latency network that interconnects the SPs and the shared memory banks. The SP is the primary thread processor in the SM. It per-

forms the fundamental floating-point operations. Each SP unit can perform fully-pipelined multiply-add operations. The SFUs units can compute transcendental functions. They also contain four floating-point multipliers.

Tesla microarchitecture has been designed for scalability, as it can be configured with a different number of units for different market segments. Moreover it features Scalable Link Interconnect (SLI), that enables multiple GPUs to act together as one, providing further scalability.

CUDA [17] is a minimal exetension of the C and C++ programming languages released by NVIDIA in 2006. It features a parallel computing platform and programming model that exploits the GPU to increase the computing performance.

A CUDA program executes serial code on the CPU and executes parallel kernel across a set of parallel threads on the GPU. The threads are organized in blocks and grids of thread blocks. Each thread within a thread block executes an instance of the kernel. A thread block is a set of concurrently executing threads that can cooperate among themselves through barrier synchronization and shared memory. A grid is an array of thread blocks that execute the same kernel, read inputs from global memory, write results to global memory, and synchronize between dependent kernel calls. This hierarchy maps to the NVIDIA architecture by assigning one or more grids to a GPU. Each SM executes one or more thread blocks and SPs execute single instructions.

CUDA platform is accessible to software developers through dedicated libraries, compiler directives and extensions to common programming languages. Moreover, it supports *OpenCL*, *OpenGL* and a number of other programming languages through third party wrappers. CUDA is currently used in HPC field to accelerate encryption, decryption and compression, to process bioinformatics algorithms, to perform medical analysis simulations, to speed-up physical computations and to perform other compute-intensive tasks.

Figure 2.2: Fermi microarchitecture.

Fermi [18] [56] [58] is a GPU microarchitecture developed by NVIDIA as the successor to the Tesla microarchitecture.

As shown in figure 2.2, Fermi architecture features an increased number of SMs, each including 32 single precision SPs, named *CUDA cores*, four SFUs, a shared memory and a cache.

The SFUs execute transcendental instructions at a rate of one instruction per thread, per clock. Therefore a warp executes over eight clocks. Each CUDA core features an integer ALU, capable of 32-bit operations. Moreover, it includes a floating point unit, that performs fused multiply-add instruction for both single and double precision arithmetic. Fused multiply-add computes multiplication and addition with a single final rounding step, with no loss of precision in the addition.

Fermi architecture provides a number of innovations with respect to the previous designs. They make Fermi more suited to HPC:

- Floating point computations in double precision take half the speed of single precision ones, versus a tenth of the speed as it is in the previous architectures;
- Error correcting codes on main memory and caches allow to detect and correct soft memory errors;
- 64-bit virtual address space allows the GPU to overcome the 4GB memory limit of the previous architecture;
- A better utilization of the hardware can be achieved thanks to fast context switching. This feature allows independent kernels to overlap their execution;
- Unified address space solves the problem of managing separate memories by placing them into a single 64-bit address space, thereby making it easier to compile and run programs on Fermi;
- Bidirectional PCIe link to the host allows to achieve a bandwidth that is four time the bandwidth of the preceding architecture.

Kepler [19] [42] is NVIDIA's first microarchitecture to focus on energy efficiency. Indeed, each Kepler core consumes less power with respect to the previous architecture. Overall performance has been increased as well by introducing a new SM design called *SMX*, that features an increased number of SPs and warp schedulers in each SM.

Figure 2.3: Kepler microarchitecture.

Kepler is also the first NVIDIA architecture designed mainly for HPC, therefore it introduces new capabilities with respect to the previous designs:

- Dynamic parallelism allows the GPU to generate work for itself and to control the scheduling of that work via dedicated hardware paths, without involving the CPU;
- Hyper-Q enables multiple CPU cores to launch work on a single GPU simultaneously, thus increasing the overall GPU utilization. Indeed, it increases the total number of work queues by allowing 32 connections to be established;
- The grid management unit manages and prioritizes grids to be executed on the GPU. This unit can suspend grids until they are ready to execute, thus providing the flexibility to enable dynamic parallelism;
- GPU direct enables GPUs within a single computer, or GPUs in different servers on the same network to directly exchange data without needing to go to CPU and system memory. Moreover, it allows third party devices to directly access memory on multiple GPUs within the system, thus freeing

the GPU DMA engines for use by other tasks.

NVIDIA has developed a family of GPU accelerators for HPC called TESLA. It features Kepler architecture combined with CUDA parallel computing model. This solution provides a new power-efficient hardware accelerator for data analytics and scientific computing applications.

The GPGPU solutions provided by Nvidia can achieve very good performances when executing intensive floating-point computations, especially those in single-precision. However, the rigid structure of the GPU-based devices causes a performance degradation and a reduction in the power efficiency of the computing system in case of low utilization. On the contrary, the solution proposed in this thesis project provides a flexible architecture, which can be configured and re-configured in order to execute each computing task in the most efficient way.

2.1.2 CPU-based Systems

The subset of the CPU-based heterogeneous systems includes a wide range of different architectures. Their common feature is the key role of the general purpose CPU, both at the architectural level and at the programming level.

IBM Power 8

IBM *Power8* [62] is a family of superscalar processors based on IBM power architecture. It has been released in 2013 and it can be implemented with a 22nm lithography. Figure 2.4 shows the structure of a Power8 silicon die.

This CPU can be equipped with up-to twelve cores, each one featuring a complex pipeline that allows eight threads to be processed simultaneously. Two or more Power8 CPUs can be connected by exploiting the integrated multi-processor link.

The multi-core chip includes a number of special accelerators *i.e.* for cryptography, virtualization and data movement, a PCIe gen 3 controller and a embedded Dynamic Random-Access Memory (eDRAM) L3 cache.

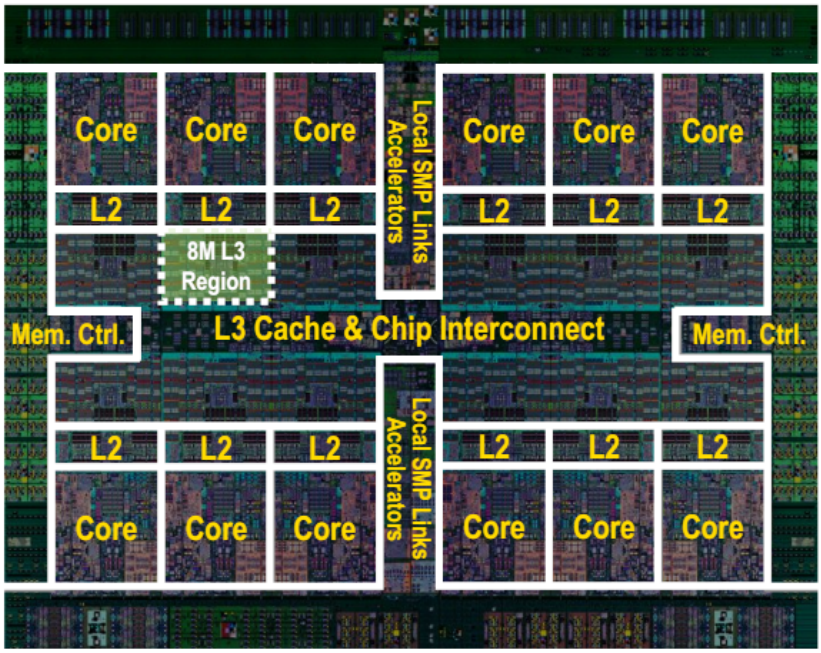


Figure 2.4: Power8 Processor Die with description.

The power consumed by the system and its temperature are managed by a dedicated micro controller, which also dynamically changes the frequency of the CPU. This controller is implemented on the same silicon die of the CPU and runs a special firmware, that manages more than a thousand Integrated Voltage Regulators (IVRs).

Power8 interfaces with the external memory through on-chip generic memory controllers paired with external components named Centaur. Centaur is a dedicated chip that acts as a memory buffer and a memory controller, moreover all the available Centaur chips can be aggregated to be used as a shared L4 cache. This solution allows to pair a Power8 CPU with different types of memory without changing its structure, only the Centaur chip needs to be changed.

Power8 has been designed for HPC industry, indeed it supports Coherent Accelerator Processor Interface (CAPI), a protocol for heterogeneous computing. CAPI [65] allows to connect an hardware accelerator to the Power8 processor via PCIe gen 3 link. The protocol is encapsulated in PCIe and takes care of the coherence between the processor and the accelerator. CAPI removes any overhead due to the OS and to the device driver and allows the accelerator to work with

the same memory addresses that the processor uses.

This hybrid solution has a simple programming paradigm while delivering high computing performance.

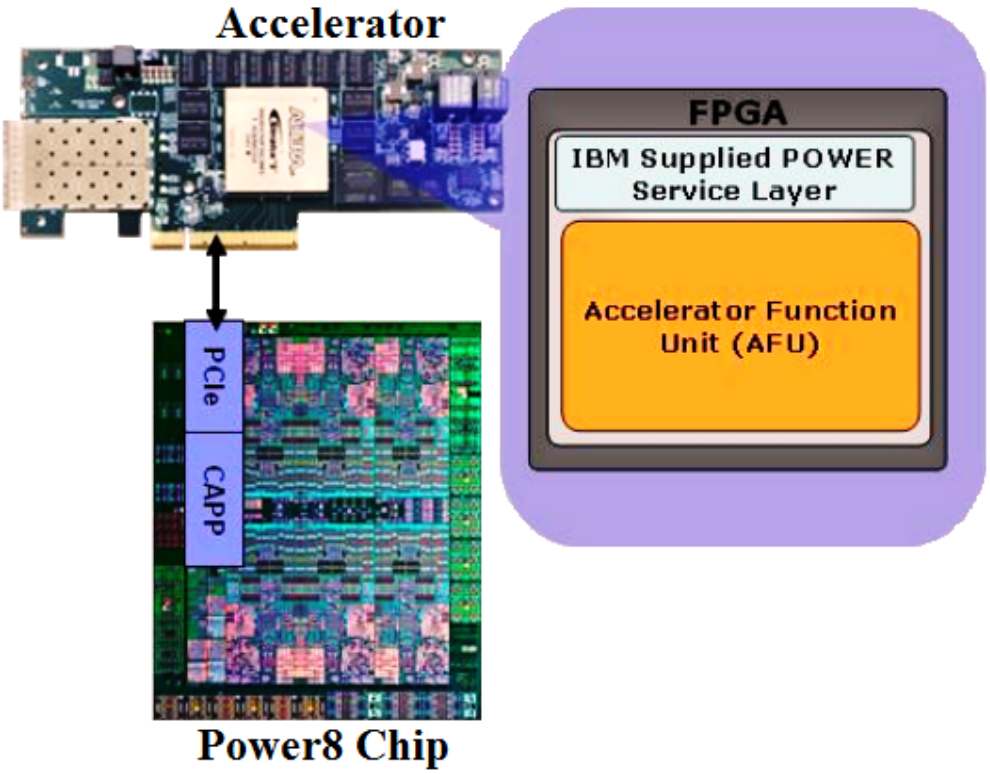


Figure 2.5: CAPI Hardware Ecosystem.

As shown in figure 2.5, the algorithm that has to be accelerated is contained in a special unit on the FPGA called Accelerator Function Unit (AFU). AFU provides applications with a high computational unit density for customized functions. Client's AFU is treated as a coherent peer by the Power8 processor, therefore data intensive programs can easily be offloaded to the FPGA, freeing the Power8 to run standard programs. On the CPU side, Coherent Accelerator Processor Proxy (CAPP), a dedicated silicon area, enables the accelerator to act as a peer to the processors on the chip.

CAPP unit maintains a directory of all cache lines held by the off-chip accelerator, therefore it acts as the proxy that maintains architectural coherence for the accelerator across its virtual memory space. Power Service Layer (PSL), a core

that resides on the FPGA alongside the AFU, works in concert with the CAPP unit across the PCIe connection.

The accelerator uses the same virtual memory space as the core application that enables it, therefore CAPP and PSL are in charge to manage all the virtual-to-physical translations, thus simplifying the programming model.

On the software side, CAPI reduces development time for algorithm implementation and allows the processor to communicate with the accelerator by eliminating all the intermediaries. Moreover, it offloads the computation-heavy functions to the accelerator, while the application executes on the host processor. Operating System (OS) kernel extensions and library functions that initialize the CAPI device and maintain the communication functions are provided with each Power8 system.

Power8 architecture and CAPI protocol deliver a high performance heterogeneous technology for HPC industry, while providing a simple programming paradigm. However this solution needs special hardware to be implemented, indeed it requires a dedicated system with one or more Power8 CPUs and at least a FPGA PCIe board compliant with CAPI protocol.

Finally only one CAPP unit is implemented on each Power8 CPU, therefore only one CAPI accelerator can be connected to a Power8 processor. IBM plans to add more CAPP units to their Power architectures in the next future.

The solution provided by IBM addresses the same challenges on which the proposed work focuses. Both the works greatly simplify the communication between the host processor and the accelerator. However, the solution provided in this thesis project allows to implement a multi-FPGA system, while IBM CAPI can manage a single accelerator only.

Intel Xeon Phi

Intel *Xeon Phi*[13][48], released in 2012, is the brand name for the intel Many Integrated Core (MIC) architecture that incorporates all the previous researches on many-core processors. It is a PCIe form factor add-in card that can be added to a Intel Xeon-based system in order to improve its performance for parallel code.

It features 61 low-speed individual cores that can execute 244 threads simultaneously, a 512-bit SIMD instruction set with wide vector units, 16GB on-board GDDR5. Up-to eight Xeon Phi boards can be included into a host server, each adding a maximum speed-up of 1.2 TFLOPS to the system.

As of June 2015, Tianhe-2, a HPC system based on Xeon Phi coprocessor, is the TOP500[28] fastest supercomputer, achieving a theoretical peak performance of about 55000 TFLOPS.

An Intel Xeon Phi-based system can be programmed using the same languages, models and tools used also with Intel Xeon processors. An application, to take full advantage of Xeon Phi, must scale over 100 threads , make extensive use of vectors and be able to exploit the high local memory bandwidth provided by the coprocessor. Each coprocessor runs a full-service Linux OS.

Xeon Phi can operate in multiple execution modes.

- Symmetric: the workload is shared between the host processor and the coprocessor;
- Native: workload resides entirely on the coprocessor;
- Offload: workload resides on the host processor and parts of the workload are sent out to the coprocessor as needed.

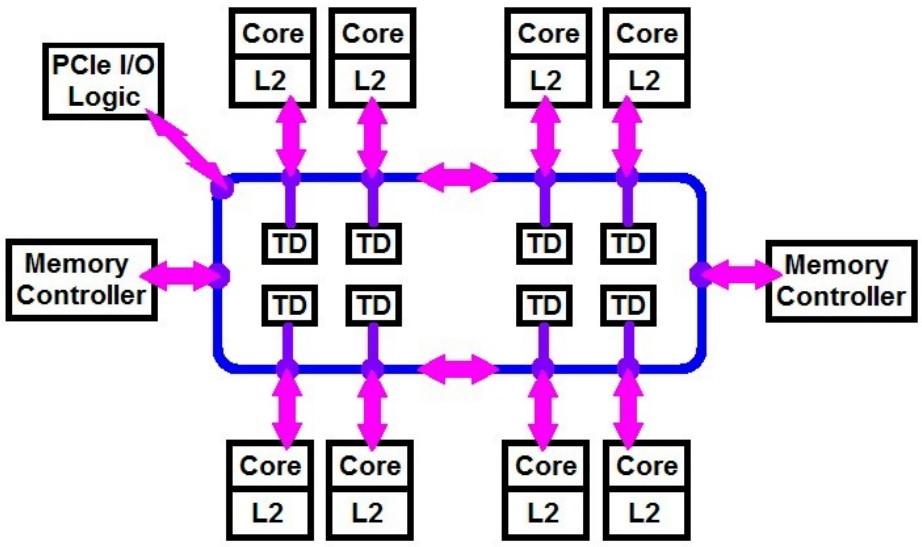


Figure 2.6: Intel Xeon Phi Coprocessor Block Diagram.

In a HPC system each Intel Phi coprocessor is connected to an host Intel Xeon processor through a PCIe bus. Since the coprocessor runs a Linux OS, it acts as an autonomous accelerator node, which can also share the execution of the applications with the host CPU. Multiple Xeon Phi coprocessors, installed in a single host system, can communicate with each other through PCIe interconnect, without any intervention of the host. Moreover, they can communicate also through a network card such as infiniband and ethernet.

As shown in figure 2.6, Xeon Phi coprocessor is primarily composed of processing cores, caches, memory controllers, PCIe client logic and bidirectional ring interconnect.

The cores in the coprocessor provide high throughput for highly parallel workloads, while featuring a high power efficiency. Moreover, they use a short in-order pipeline and support four threads in hardware. Each core includes a Vector Processing Unit (VPU) that features a 512-bit SIMD instruction set. Therefore it can execute 16 single-precision or 8 double-precision operations per cycle. Moreover, the VPU supports fused multiply-add instructions that can execute 32 single-precision or 16 double-precision floating point operations per cycle.

The VPU also features an extended math unit that can execute transcendental operations, thereby allowing these operations to be executed in a vector fashion with high bandwidth. The extended math unit operates by calculating the polynomial approximations of the transcendental functions.

Each core has a L2 cache that is kept coherent by a global-distributed tag directory (TD). When an L2 cache miss occurs on a core, the core generates an address request on the address ring and queries the global-distributed tag directory. If the data is not found in the tag directory, the core generates another address request and queries the on-board memory for the data. Once the memory controller retrieves the data block from memory, it is returned back to the core over the data ring.

The on-chip ring interconnect, which is implemented as a bidirectional link, links all the components with the interface to the on-board GDDR5 memory and with the PCIe client logic. Each direction of the link is composed of three inde-

pendent rings: a 64 bytes-wide data block ring which supports the high bandwidth requirement due to the large number of cores, a smaller address ring used to send read/write commands and memory addresses and an even smaller acknowledgement ring, which sends flow control and coherence messages.

Xeon Phi coprocessor can be put in a power-saving mode when is not being used. As soon as all the four threads on a core are halted, the clock to the core is gated. After a programmable time, the core power gates itself. When all the cores of the device are power gated, the tag directory, the interconnect, the L2 caches and the memory controllers are clock gated. At this point, the host system puts the coprocessor in an idle state, wherein all the cores are power gated, the GDDR5 is kept into a self-refresh mode and the PCIe logic is put in a wait state. These power management techniques make Xeon Phi coprocessor suitable for HPC data centres.

Both the work proposed in this thesis project and the Intel Xeon Phi architecture provide a high-performance power-efficient supercomputing system composed of a chain of PCIe accelerator boards. However, they address the need for flexibility by providing two different solutions: Xeon Phi is based on a number of interconnected general purpose processors, while the proposed architecture exploits FPGA technology.

Intel Stellarton

Intel *Stellarton* [10] [40] [41] is a System on a Chip (SoC) architecture composed of a general purpose processor paired with a FPGA. Stellarton has been released in 2011 by Intel as the Atom E600C series, which is the first configurable Intel Atom-based processor, featuring a 45nm lithography process and a Thermal Design Power (TDP) of 7W.

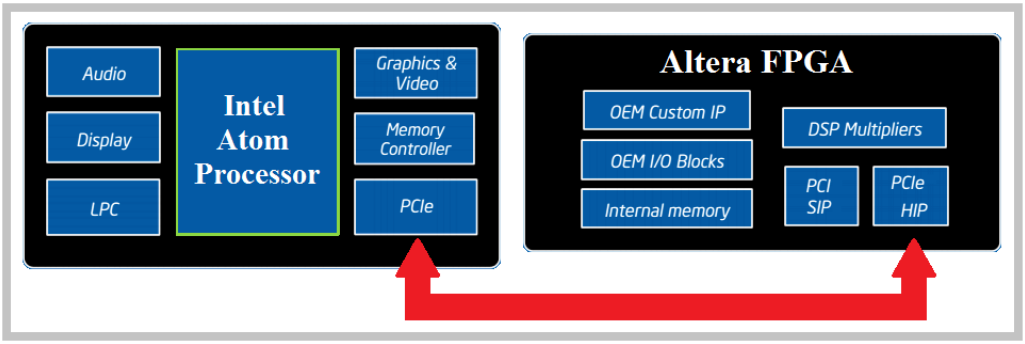


Figure 2.7: Intel Stellarton architecture.

As shown in figure 2.7, Intel Stellarton includes a Atom E600-based processor paired with an Altera FPGA in a single package, multi-chip device suited to integrated solutions.

On the CPU side, the SoC provides a Intel Atom x86 processor featuring Intel Hyper Threading technology and integrated hardware-assisted Intel Virtualization technology for 32-bit Intel architecture. The processing system includes also a Intel graphics media accelerator, a memory controller for up to 2GB DDR2 memory, an audio controller and a PCIe interface.

The included FPGA is an Altera high-performance, power-optimized device, that features 6 high-speed transceivers, a number of Digital Signal Processing (DSP) blocks and a PCIe hard IP. The FPGA is compliant with Altera Quartus II software, that allows to design custom Intellectual Properties (IPs) and to purchase third party soft IPs from Altera partners.

Although the CPU and the FPGA belong to the same chip, they act as they were separated devices. Indeed, they are built on two different silicon dies, connected only through a PCIe x1 gen1 link. This solution is the weak point of the Stellarton architecture for two main reasons. First, the one-lane PCIe link is a bottleneck for the entire system, because it is the only interconnection between the FPGA and the CPU and it does not allow to achieve a sufficient throughput. The PCIe link was already too slow in the 2010, when Stellarton has been designed. Moreover, the adopted solution needs two PCIe interfaces to be implemented: one on the CPU side and the other on the FPGA side. On the CPU side, there is only a PCIe interface, therefore no external PCIe peripherals can be connected to

the CPU. On the FPGA side, there is only a PCIe hard-IP, therefore, in order to connect extra PCIe peripherals to the FPGA and to the system, additional third-party soft IPs must be purchased.

However, Intel Stellarton provides a number of capabilities, mainly focused on the flexibility of the architecture and on the reduction of the design effort: the single package combining multiple functions allows to reduce the footprint of the designs and the cost of the materials. Moreover, multiple I/O configurations can be implemented with the same device. The configurable FPGA allows to integrate proprietary features into the the SoC package and to change the platform details later in the design cycle, thus saving resources and money.

Intel Stellarton supports Microsoft embedded Windows versions, Meego Linux and VxWORKS OSs.It is suitable for a variety of embedded applications, such as industrial machines, portable medical equipment, communication gear and vision systems.

Although it is not intended for the HPC field, Intel Stellarton provides an interesting coupling between a general purpose processor and a FPGA device. As in the architecture proposed in this thesis project, the CPU and the FPGA communicate through PCIe connection.

Xilinx Zynq

Xilinx *Zynq* is a family of SoCs developed for high-end embedded-systems applications, such as video surveillance, automotive-driver assistance, next generation wireless and factory automation. These devices combine a ARM processing system, which is capable of running a variety of OSs, with a Xilinx FPGA, that interfaces with the processing system.

Vivado Design Suite allows to design a system including both the ARM CPU, both the custom hardware programmed on the FPGA. Moreover, using *Vivado HLS*, developers can implement custom hardware accelerators starting from their C language specification.

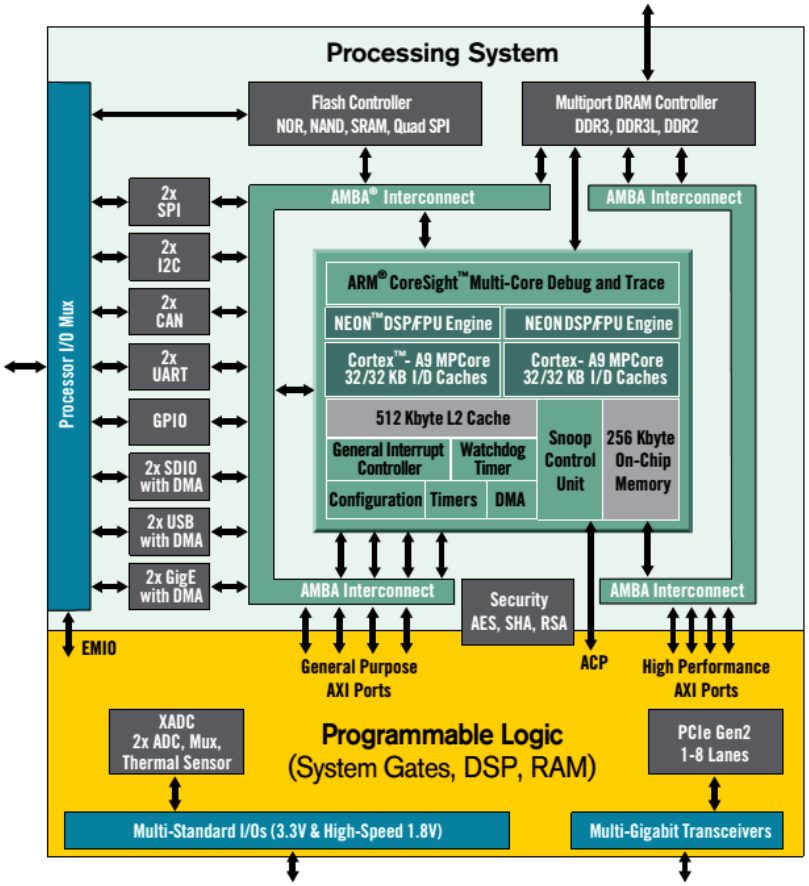


Figure 2.8: Zynq 7000 block diagram.

Zynq-7000[46][38], released in 2012, is the first version of the Zynq family. It consists of a fully programmable platform: custom Software (SW) can be executed on its ARM CPUs, hardware accelerators can be implemented on the FPGA and all its interfaces can be programmed. This capability can be achieved thanks to the thigh integration between CPUs, FPGA, DSP and mixed signal functionality.

As shown in the upper part of figure 2.8, Zynq-7000 features a complex Processor System (PS) composed of a 1GHz dual core hardened ARM Cortex A9 CPU, on-chip memory, Random Access Memory (RAM), flash memory controllers and AXI peripheral blocks. The fast memory controller supports 1333MHz and 1866MHz Synchronous Dynamic Random-access Memories (SDRAMs) with or without Error-Correcting Code (ECC). More memory controllers can be placed in the Programmable Logic (PL) as soft IPs.

The PL, which is painted in yellow, is connected to the PS through nine AXI interfaces that allow to exploit all the available bandwidth between the PS and the PL. Therefore, several separate hardware accelerators, implemented in the PL, can be simultaneously connected to the PS. Zynq-7000 PL is based on *Kintex-7* and *Artix-7* FPGA technology, that allows the designers to port Register-Transfer Level (RTL) code and IP blocks from Kintex-7 and Artix-7 FPGAs without modifications or optimizations.

Thanks to the ARM Cortex CPU, the Zynq platform can run a wide range of OSs, spanning a number of applications. Moreover, custom C code can be executed directly in the ARM CPU. If the implementation is not fast enough, the designers can implement a Hardware (HW) version of the algorithm using Vivado HLS and then program it on the PL.

Zynq-7000 can be configured to perform a security procedure at startup. Indeed, the CPU, which boots before the FPGA, can perform a boot sequence supporting user authentication, encryption and data authentication. After the boot sequence, the authenticated and decrypted code is placed into the on-chip memory, where it executes. Moreover, each Zynq device can monitor its environment by exploiting external analog sensors connected to its interfaces in order to detect fails and intrusions. If the device detects an intrusion, it can clean its memory and its PL, thus avoiding reverse-engineering and manumissions.

Zynq UltraScale+[39], released in 2015, is the new version of the Zynq family. It provides five times more performance per Watt with respect to the Zynq-7000 family.

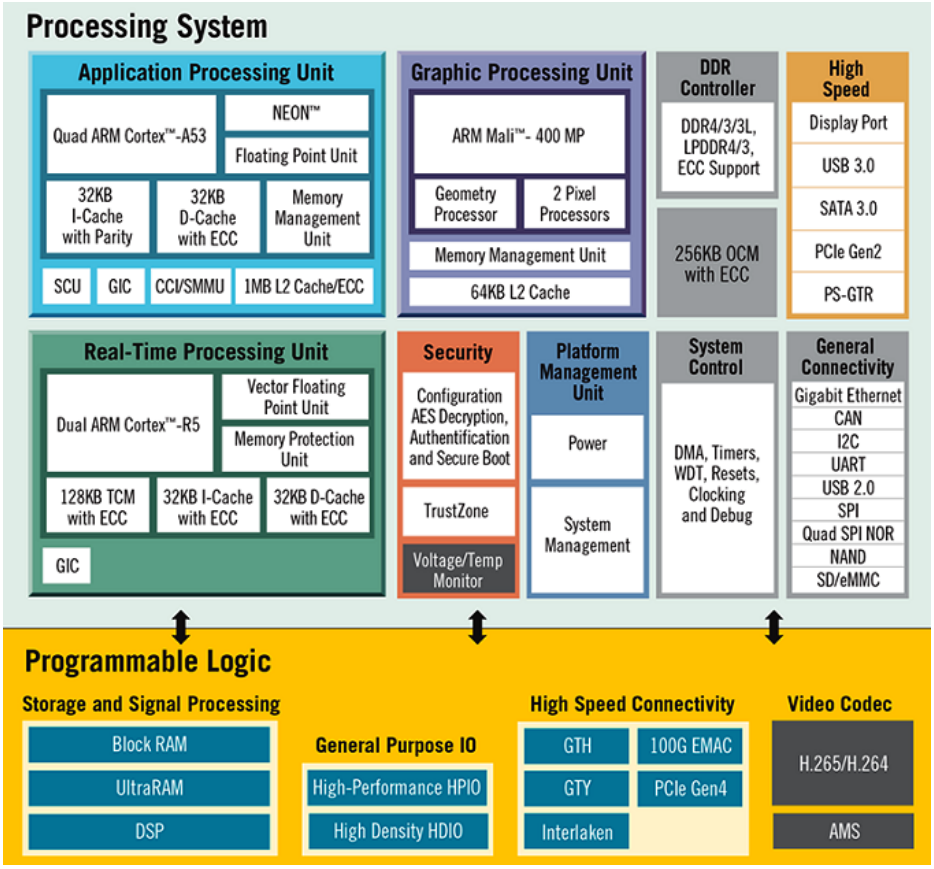


Figure 2.9: Zynq Ultrascale block diagram.

As shown in figure 2.9, the architecture is composed of four 64-bit ARM Cortex A53 cores, running at 1.3GHz, two 32-bit ARM Cortex R5 real-time Micro Controller Unit (MCU) cores, running at 600MHz, a ARM Mali-400MP GPU and an Ultrascale FPGA fabric manufactured with a 16nm FinFet process. This device supports both DDR3 and DDR4 memories, and allows to use the ECC capability. Zynq Ultrascale+ provides a number of high-speed interfaces, such as Universal Serial Bus (USB) 3m Peripheral Component Interconnect Express (PCIe) gen4, gen3 and gen2, interlaken, ethernet and serial transceivers.

The overall power efficiency is ensured by multiple power domains and power gated islands. Moreover, the designers can use each core for the task that best fits its capabilities: an OS can execute on the quad-core CPU while assigning its real-time tasks to the dual-core CPU. The graphic workload can be managed by the dedicated GPU and all the other tasks can be accelerated in-hardware on the PL.

This approach increases the flexibility of the platform, while leveraging its performance and its power efficiency.

Although Zynq platform is primarily intended for embedded applications, it provides an innovative approach to the use of programmable logic paired with general purpose processors. Indeed, a FPGA device and a multi-core CPU are coupled on the same silicon die. This technique could be a promising solution for the future HPC systems.

2.1.3 FPGA-centric Systems

FPGA-centric systems include a number of flexible and efficient architectures for the HPC field, in which the FPGAs play a key role. Indeed, these architecture feature a great number of FPGA-based accelerators included into a single server-class computing device. The accelerators ensure all the processing power required by the applications, while the CPU manages the whole architecture.

Pico Computing

Pico Computing[21] provides a broad range of HPC solutions both for datacenter and desktop applications.

Pico Computing's scalable architecture exploits FPGA technology to enable high performance compute density, energy efficiency and simplified application design by packing many large-scale FPGAs per PCIe slot and providing direct FPGA-to-FPGA communication and data passage without host involvement.

The building blocks of the architecture are the modules: business card-sized computing elements composed of a high-performance FPGA logic, a local memory subsystem and a fully-switched PCIe Eight-Lane (x8) communication structure. The various modules provide options in FPGA choice, memory configuration, I/O and other features. Several modules can be used in standalone embedded applications, as they are fully integrated with embedded Linux OS. The latest module features a Xilinx Kintex Ultrascale FPGA coupled with a 4GB Hybrid Memory Cube (HMC). Moreover, it provides a PCIe x8 gen3 interface for backplane connectivity and OPENCL support for software development.

Pico Computing provides a variety of PCIe backplanes that can host up to six modules on a full-length PCIe card. Each module can be exchanged or upgraded by snapping it to the backplane in a plug-and-play fashion. Each backplane is based on a fully-switched PCIe x16 gen3 bus from the host computer and provides independent PCIe x8 gen3 buses to each module. This allows the host to directly address up to 6 FPGAs per board and up to 48 FPGAs in a single 4U chassis.

A chassis unit is a compute, standalone server class rack mount system that features a dual quad-core Intel Xeon CPU, up to 192GB DDR3 RAM memory, eight PCIe x16 gen3 interfaces and two 10Gbps ethernet ports. The host system, which runs Linux OS, exploits Pico computing's software framework to control the applications running on the FPGA modules within a chassis.

The Pico Computing framework is a Linux-based design utility that provides the link between the application software running on the host computer and the HW algorithm implemented in the FPGA. It manages the board-level implementation of FPGA designs, data flow, memory, system communication, monitor and debug. The framework frees the designers to focus only on the details of the application to be developed and occupies about the 1% of the overall FPGA area. Moreover, it allows to load the configuration bitfiles to the modules, to move the data from software into the FPGAs and to configure the FPGA communication in a number of different ways. At the software level, the framework allows to manage the communication within a 48-FPGAs architecture and to monitor FPGA's temperature, current and voltage to ensure the proper behaviour of the system.

Pico Computing provides also a full-heigh, GPU-length, PCIe board featuring up to eight HMC devices and a single high-performance Xilinx Ultrascale FPGA, called Single-board Supercomputer. The on-board HMCs can be configured as one x16 lane with up to four HMCs chained or as two x8 lane with two HMCs accessed independently or chained. The Single-board Supercomputer allows to implement high-bandwidth accelerated applications, thanks to the on-board HMC memory. Moreover, it can be scaled by configuring a single chassis with up to eight boards.

The architecture illustrated in this thesis project is similar to the one provided by Pico Computing, as both are based on a multi-FPGA technology. Pico Computing exploits the high performance PCIe host bus to implement the board-board communication and to manage the FPGA devices carried by each board. On the contrary, the proposed architecture exploits a dedicated high-performance board-board link, while using the PCIe bus for host-board communication only.

Convey Computer

Convey Hybrid-core computers[4][64] tightly integrate an FPGA-based, reconfigurable coprocessor with an Intel processor in a standard rack-mountable enclosure. This solution offers enhanced performance without sacrificing the flexibility and ease of use of a general-purpose machine.

The host system runs industry-standard Linux and supports standard networking and interconnect fabrics. The coprocessor's FPGA executes specific operations, called *personalities*, that represent a large component of application's runtime. Convey provides a set of personalities for key application areas in industries like life sciences, governmental programs and big data analytics.

An application executable contains both Intel and coprocessor instructions, that execute in a common, coherent address space. Therefore, coprocessor instructions can be thought of as extensions to the x86 instruction set. The personalities include a base set of instructions that are common to all personalities. Moreover, they also include a set of extended instructions that are designed for a particular workload.

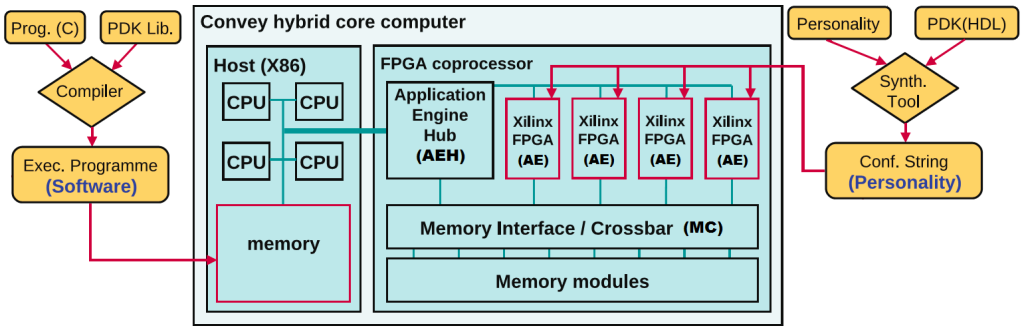


Figure 2.10: Convey Hybrid-core architecture and design flow.

As shown in figure 2.10, the coprocessor comprises a Application Engine Hub (AEH), a set of Application Engines (AEs) and Memory Controllers (MCs).

The AEH acts as a control hub for the coprocessor. It provides the Hybrid-Core Memory Interconnect (HCMI) to the host processors, fetches and decodes instructions and executes Convey base instructions, while passing extended instructions to the AEs for execution. Eight MCs support a total of 16 memory channels, that provide a highly parallel and high bandwidth connection between the AEs and the coprocessor memory. The AEs implement the extended instructions that deliver performance for a personality. They are connected to the AEH by a command bus that transfers instructions and scalar operands and linked via a network to each of the MCs.

The host system runs a customized CentOS Linux OS that supports Convey architecture. Applications can access the coprocessor either by linking in Convey math libraries with coprocessor versions of common algorithms or by recompiling with Convey compilers, that can generate both Intel64 and coprocessor instructions form C/C++ or Fortran.

A personality is the custom logic that resides on the coprocessor and implements the extended instruction set designed to accelerate a given algorithm. Each personality includes a pre-compiled FPGA bit file, a description of the machine state model that allows the compiler to generate and schedule instructions and an ID used by the application to load the correct image at runtime. A system can contain multiple personalities that can be dynamically loaded, but only one personality is loaded at any one time.

Convey offers also a family of PCIe form factor coprocessors featuring the latest high-density Xilinx FPGAs, that can be installed in standard x86 servers as hardware accelerators for key algorithms. The PCIe coprocessors support all the capabilities provided by Convey to its Hybrid-core computers, such as dynamic reconfiguration, personalities and the common address space. Moreover, they can be managed, configured and customized using the Convey tool set. Each accelerator occupies a full-length, double-height PCIe slot and can be configured with FPGAs of different densities and multiple power and cooling solutions.

Moreover, it can operate as a memory-free accelerator with no local storage or with up to 64GB on-board DDR3 memory.

Convey Hybrid-core computers provide a multi-FPGA computing system hosted by a standard rack module. This solution is similar to the one provided by this thesis project. However, Convey implements a fully-custom hardware architecture, while the proposed design is composed of a standard Personal Computer (PC) and a number of commercially-available FPGA boards.

Maxeler

Maxeler Architecture [14] employs one or more FPGA boards, called DataFlow Engines (DFEs), to accelerate compute-intensive algorithms. A single DFE acts a FPGA-based accelerator, that works jointly with the host processor. Maxeler Architecture will be discussed in detail in section 2.3.

2.1.4 Custom Systems

Custom HPC systems can achieve extremely high performance when processing the small set of algorithms for which they have been designed. These systems are based on custom Application-Specific Integrated Circuit (ASIC) processing units, that provide a huge efficiency at the cost of a limited flexibility.

Anton Machine

Anton Machine is a special purpose, massively parallel supercomputer for biomolecular simulation, designed and constructed by D. E. Research. Its main purpose is to increase the speed of molecular dynamics simulations, compared with the previous state of the art.

An Anton machine consists of a substantial number of ASICs, interconnected by a specialized high-speed, three dimensional torus network. Two Anton machines have been designed and constructed, both as 512 nodes supercomputers: Anton1 and its successor, Anton2.

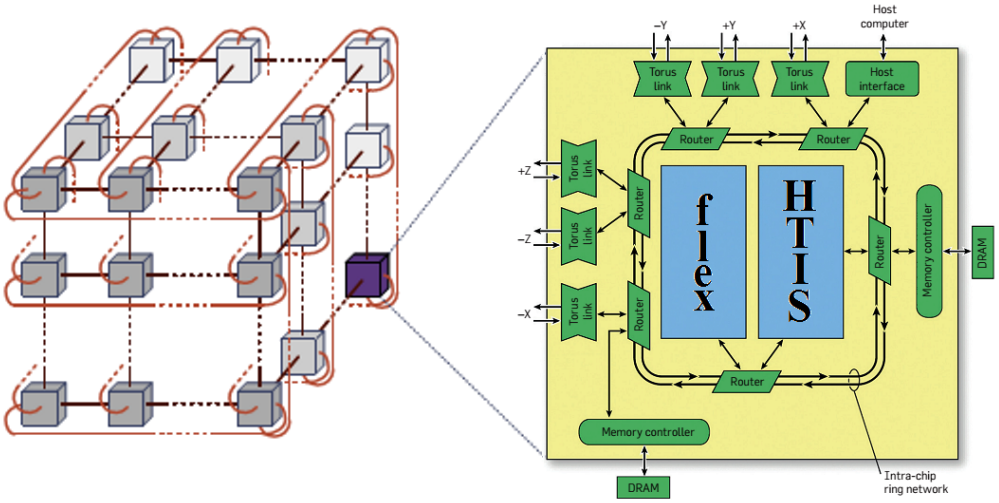


Figure 2.11: Anton1 architecture.

Anton1[51], designed in 2008, is the first version of the Anton machine family. The machine is based on a number of identical specific ASICs that interact in a tightly coupled manner using a three dimensional torus network. Each torus link provides 5.3GB/s full-duplex communication with a hop latency of 50ns.

A block diagram of the network is depicted in figure 2.11, which also shows the structure of an Anton1 ASIC. The building block of the system is a node, that comprises an Anton1 ASIC, attached DRAM and six ports to the system-wide interconnection network. Four nodes are implemented on the same Printed circuit board (PCB), thus constituting a node board, which fits in a rack.

The main components of the Anton1 ASIC are the HTIS and the Flex. High-Throughput Interaction Subsystem (HTIS) is a highly specialized, largely hard-wired unit that takes care of most of the computing workload.

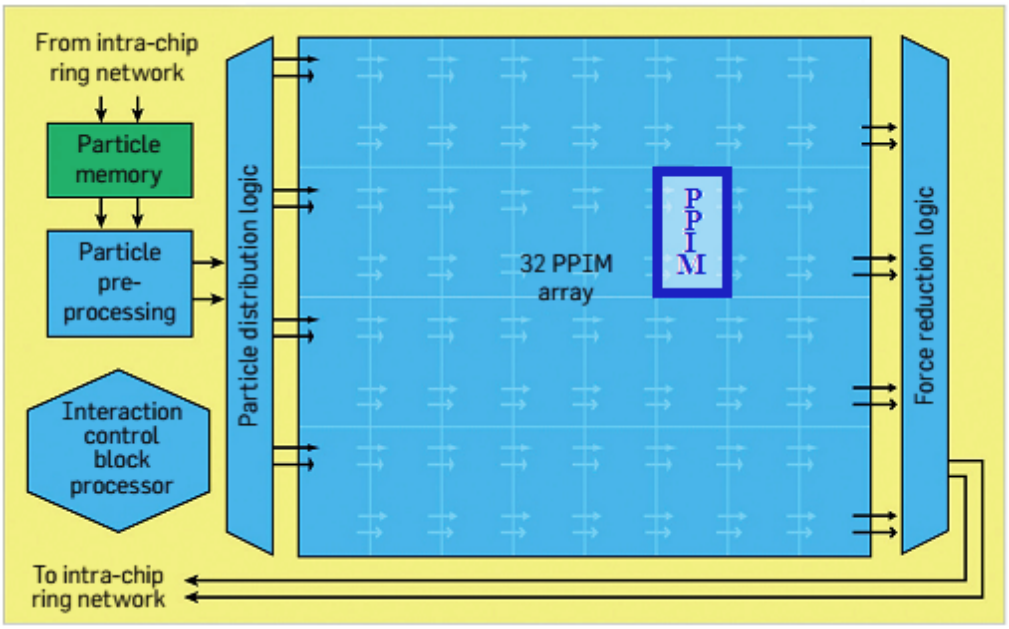


Figure 2.12: Anton1 HTIS tile detail.

As figure 2.12 shows, each HTIS contains 32 deeply-pipelined Pairwise Point Interaction Modules (PPIMs), organized as four chains of eight PPIMs each and an Interaction Control Block (ICB).

Each PPIM includes a 26-stage pipeline running at 800MHz, featuring a data path with variable numerical precision. This solution allows to save time and resources, still producing a 32-bit fixed point result. The PPIM is the most hard-wired component of the architecture, however some of its features can be customize by programming special SRAM Look-Up Tables (LUTs), whose contents are determined at runtime.

The ICB is composed of two communication ring interfaces, a large buffer area and an embedded processor core, which controls the flow of data through the HTIS and acts as a buffering, prefetching, synchronization and writeback controller.

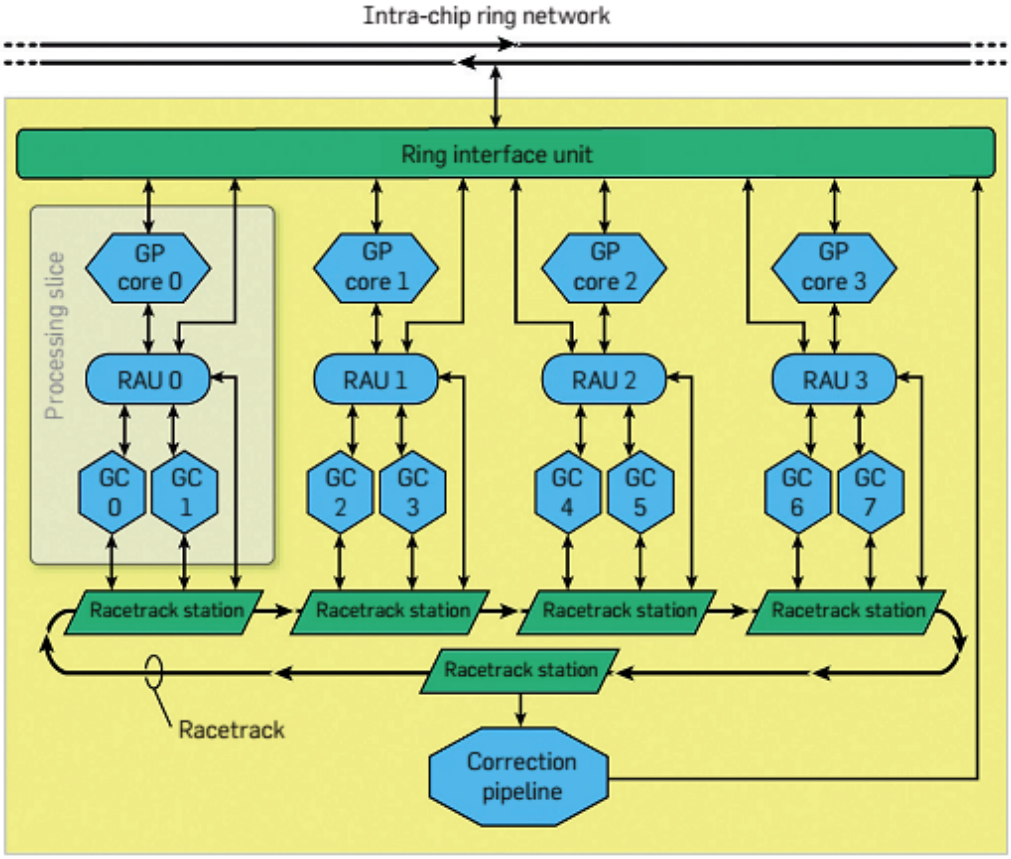


Figure 2.13: Anton1 flex tile detail.

Flexible Subsystem (Flex) is a programmable, but still specialized unit running at 400MHz, which executes the computations that can not be performed on the HTIS unit, therefore it allows the entire system to adapt to the changes made to the algorithms. Flex unit handles a variety of tasks, some involving calculation and others involving system management and maintenance functions. It also performs boot, diagnostics, self-test, loading simulations, switching contexts, logging, checkpointing and error reporting.

Referring to figure 2.13, each Flex unit is composed of four identical processing slices. A slice comprises a general purpose core with its caches, a Remote Access Unit (RAU) and two Geometry Cores (GCs).

The general purpose cores manage the data transfers and perform critical synchronizations. Each core also connects to a 32KB scratchpad memory in the core's attached RAU. The scratchpad memory is used to stage the simulation data for

background transfer by the RAU. The general purpose cores also handle all the maintenance tasks.

The RAU is a programmable data transfer engine, that includes a scratchpad memory and an array of transfer descriptors. Once the general purpose core has initialized a transfer descriptor, the RAU takes over and performs the transfer itself, freeing the general purpose core to perform other tasks. The RAU implements 128 transfer descriptors, allowing multiple concurrent transfers.

GCs perform most of the Flex's computation. Each GC is a dual-issue, statically scheduled SIMD processor with pipelined multiply-accumulate support. The GC's Instruction Set Architecture (ISA) includes vector operations and scalar operations performed by accessing the SIMD register file as a large scalar register file.

The communication subsystem provides high-speed, low latency communication bith between ASICs and among the subsystems between an ASIC. Within a chip, two 256-bit communication rings link all subsystems and the six inter-chip torus ports. Routing is performed using a global 48-bit address space, with 16 bits of node identifier and 32 bits of address per node. The communication subsystem also allows access to an external host computer system for input and output of simulation data.

A working Anton1 machine segment, composed of 128 boards, allows to achieve a speed-up of 100x with respect to a supercomputer implemented with general purpose processors, when processing molecular dynamics algorithms. Each board comprises 4 ASIC nodes and their dedicated DDR2 RAM memory. Therefore, the segment features 512 ASIC nodes, 2560 Tensilica general purpose processors, 4096 GCs and 16896 PPIMs for special-purpose computing. Each ASIC consumes 75W and the total power consumption of the system, also including memory, power supply and cooling is about 116,5KW.

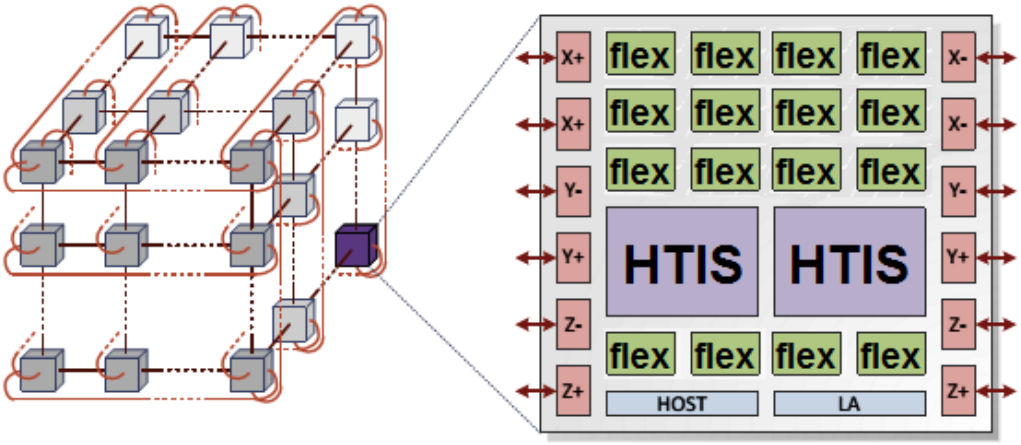


Figure 2.14: Anton2 architecture.

Anton2[52], designed in 2014, is the second version of the Anton family. It is the successor of the Anton1 machine. Like its predecessor, Anton2 comprises a number of identical ASIC nodes that are directly connected to form a three dimensional torus topology. Figure 2.14 shows the structure of the network and the block diagram of a node. Inter-node connections consist of two bidirectional channels, providing a total bandwidth of 28GB/s.

Almost all the area of each ASIC node has been dedicated to the computation, indeed the number of Flex units has increased from one to sixteen with respect to Anton1 ASIC and the number of HTIS tiles has been doubled.

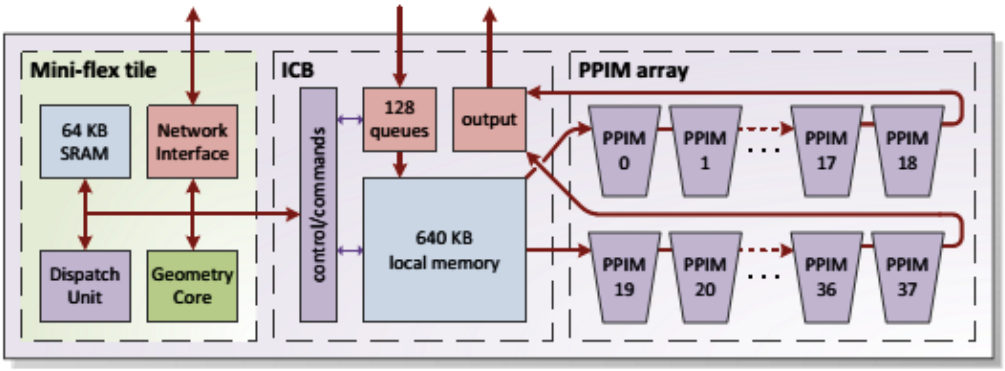


Figure 2.15: Anton2 HTIS tile detail.

As shown in figure 2.15, the Anton2 HTIS tile is similar to the corresponding unit of the preceding machine, except for the mini-Flex tile included in the

design, which is a scaled-down version of the Flex unit. This solution increases the flexibility of the entire system. Indeed, thanks to the mini-Flex tile, the hard-wired HTIS unit can be adapted to the changes made to the algorithms during the life of the machine. Moreover, by controlling the ICB with a mini-flex tile, the programmers can use the same tool chain for all embedded software and the code can be shared between the Flex tile and the HTIS tile software.

The Flex unit has been completely re-designed and its structure ha been simplified. Indeed, it comprises four GCs embedded processors, a dispatch unit that provides hardware support for fine-grained event-driven computation, 256KB of SRAM and a network interface. Its structure is illustrated in figure 2.16.

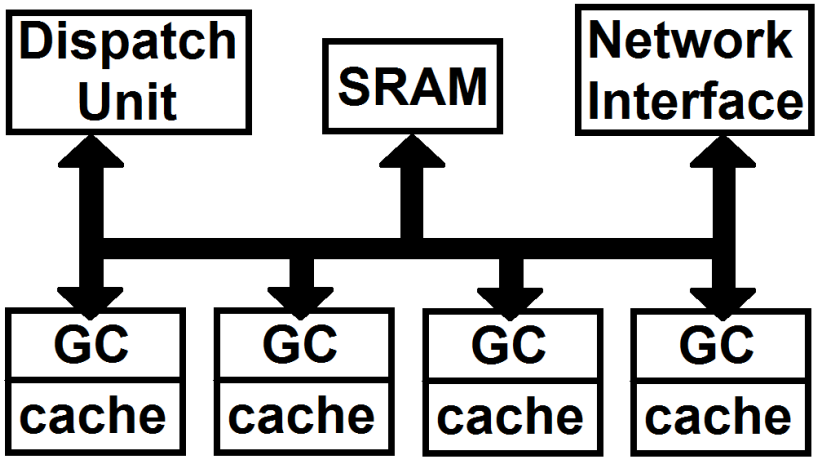


Figure 2.16: Anton2 flex tile detail.

The Anton2 ASIC architecture is clocked at 1.65GHz and the on-chip mesh network is compose of 40GB/s bidirectional links. An external host processor can be connected to the system over a PCIe link.

Anton2 features a number of improvements over Anton1 machine. The three main changes deal with the architecture of the system and with its programming model.

Anton1 has a coarse grained architecture: all data must arrive at a tile before the computation begins and all computations must be completed before the output data is sent. Therefore, there is no overlap between the different phases. On the contrary, Anton2's fine-grained event-driven implementation communicates

the results as soon as they have been computed, and individual computations begin as soon as their data is available. This enables significant overlap between the phases.

Anton1's PPIM is composed of two separated pipelines that perform different computations on the same cycle. Anton2's PPIM replaces the heterogeneous pipelines with two identical pipelines that can operate independently on different data.

Anton1 features three distinct types of embedded processors, which are programmed using a mix of C and assembly language and must be carefully coordinated with synchronization protocols. Anton2 has a single type of embedded processor, which is programmed in C++.

A working prototype of the Anton2 machine, which comprises 512 ASIC nodes, is 180 time faster than any other comparable platform with respect to the peak performances.

On large chemical systems Anton2 has a performance gain of about ten times over a Anton1 machine with the same number of nodes. Additional performance could be obtained on larger Anton2 machines. Indeed, the architecture scales-up to 4096 nodes.

Although Anton Machine has been implemented by following a completely different approach with respect to the one proposed in this thesis project, it provides an example of a fully-customized HPC architecture, which achieves a huge performance, at the cost of a extremely low flexibility.

2.2 Multi-FPGA architectures

This section deals with multi-FPGA architectures, composed of a huge number of FPGA-based processing nodes linked with a network infrastructure. These complex systems have been implemented by exploiting the scalability of the architecture, which is ensured by the high power efficiency provided by the FPGA technology.

Microsoft Catapult

Catapult[15][60] is a Microsoft Research project that aims to exploit FPGAs to improve the performance, reduce power consumption and add new capabilities to the datacenters.

A Catapult architecture, which comprises a custom FPGA board, has been implemented and used to improve the performance of Bing's search engine. However, thanks to the flexibility of the architecture, many other applications and services can be accelerated as well.

Catapult infrastructure is based on a Microsoft datacenter, that is composed of more than 1600 half-rack machines. A server features two Xeon CPUs, 64GB DRAM, two Solid State Drives (SSDs), four Hard Disk Drives (HDDs) and a 10Gb network card. Catapult fabric is embedded into each cluster node in the form of a small FPGA daughter card connected to the server via PCIe interface. All the 48 FPGA boards in a rack are directly connected through a low latency dedicated link, that is configured as a 6x8 2D torus network. This distributed solution has been adopted to allow the SW services running on the datacenter to allocate groups of FPGAs to provide the necessary area to implement the desired functionalities. Moreover, the reconfigurable fabric of the Catapult architecture allows the SW services to dynamically change the allocation of the resources in the datacenter.

On top of the architecture, Catapult incorporates a complex managing system, which takes care of the correct functioning of the infrastructure, recovers from failures and collects debug information.

The Catapult daughter board features a Altera Stratix V FPGA paired with a 8GB DDR3 memory with enabled ECC support. A 32MB on-board flash memory holds the FPGA configurations. The card interfaces with the server mainboard using PCIe gen3 x8 protocol via a mezzanine connector, which also powers the device. Moreover, a mini-SAS network connection allows the FPGA board to communicate with the other boards of the same rack. Microsoft has developed a custom PCIe interface for the Catapult FPGA device, that allows to achieve very low latency transfers between the host server and the FPGA board. More-

over, it features Direct Memory Access (DMA) support and is safe for multi-threading. Multi-threading support is achieved also by dividing the input and output buffers of the Catapult device into slots with status bits and by assigning each thread to a slot.

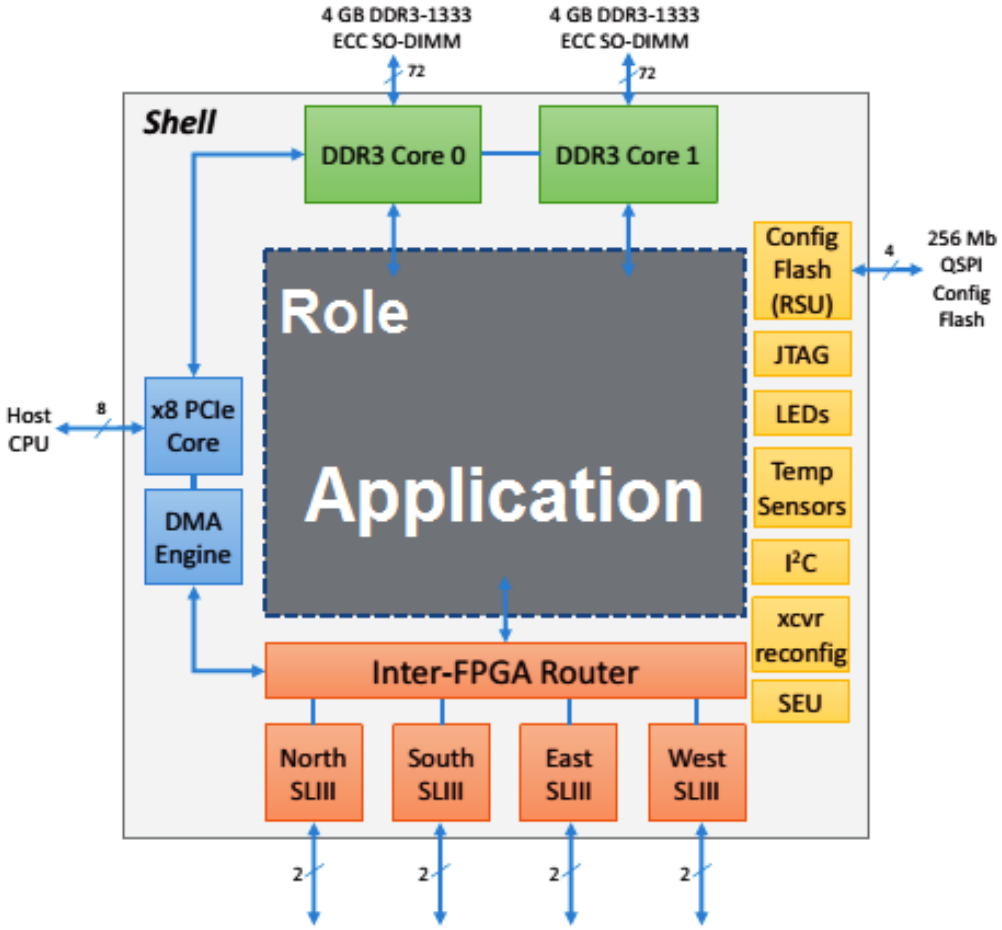


Figure 2.17: Block diagram of the Catapult FPGA architecture.

As shown in figure 2.17, the programmable logic on the FPGA is divided into two parts: the shell and the role. The shell is the reusable portion of the programmable logic, common across applications. It handles all the I/O and management tasks and exposes simple FIFOs to the role unit.

The shell unit is composed of a variety of HW cores:

- two DRAM controllers, which can be operated independently or as a unified interface;

- four high-speed SLIII links that provide a lightweight protocol for communicating with neighbouring FPGAs;
- a inter-FPGA router, which manages the traffic arriving from the PCIe interface, the role unit or the SLIII cores;
- a reconfiguration logic to read/write the configuration flash memory;
- a PCIe core with DMA support.

The shell consumes about one-fourth of each FPGA resources, although extra capacity can be obtained by discarding unused functions.

The role unit contains the application logic itself, restricted to a large fixed region of the FPGA chip. The role can access the interfaces and the capabilities provided by the shell. Moreover, it supports partial reconfiguration, which allows for dynamic switching between different roles while the shell remains active.

At the software level, the correct operation of the system is ensured by a variety of techniques developed within the Catapult project. They mainly controls the reconfiguration phase, that can cause instability to the system. In order to prevent the system to be destabilized, the driver that manages the reconfiguration of the FPGA disables the non-maskable interrupts for the PCIe device during reconfiguration. Moreover, each device sends a "TX HALT" message to its neighbouring devices to avoid the corruption of the other FPGAs. When a FPGA comes out of reconfiguration, it transmits a "RX HALT" message to its neighbours and throws away all the messages coming from them. Then, the software that manages the infrastructure tells each server to release the "RX HALT" message once all the FPGAs have been reconfigured.

Catapult infrastructure keeps monitoring several parameters of the architecture in order to detect both SW and HW errors. If a failure is detected, the SW manager can automatically reconfigure the FPGAs on the failing nodes. When the reconfiguration does not help in solving the problem, the manager reboots the system and asks for manual service if needed.

Although the architecture proposed in this thesis project provides a multi-FPGA system to be included into a single computing node, it is similar to Cata-

pult as it is intended to be scaled up to a complete datacenter by connecting more FPGA-accelerated nodes together.

NetFPGA SUME

The *NetFPGA Project*[16] is an effort to develop open source hardware and software for rapid prototyping of computer network devices started in 2007 as a research project at Stanford university. The work is based on a FPGA networking platform, which allows to develop custom HW in an open source environment. From 2007, different versions of the platform have been released.

The current version, called *NetFPGA SUME*[68], is a low-cost FPGA-based PCIe board with I/O capabilities for 100Gbps operation as a network interface card, multiport switch or test and measurement environment that enables rapid prototyping of 10Gbps and 40Gbps applications and allows to carry on projects focusing on 100Gbps applications.

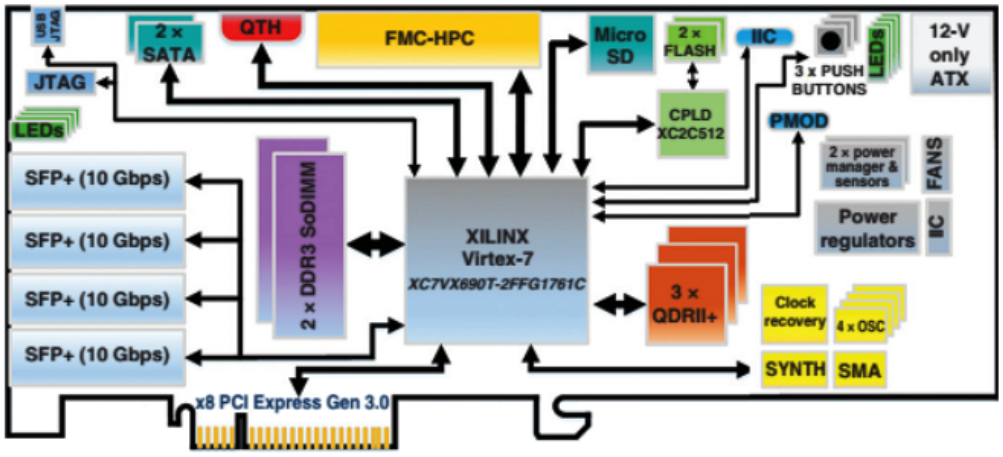


Figure 2.18: NetFPGA SUME board architecture.

The NetFPGA SUME board, which is outlined in figure 2.18, features a large Xilinx Virtex7 FPGA that supports high-speed serial interfaces, a large extensible quantity of high-speed DRAM and a high-throughput SRAM. The FPGA provides 30 serial links connected to GTH transceivers, that run at up to 13.1Gbps. These links connect four standard 10Gbps SFP+ ethernet interfaces, two expansion connectors and a PCIe connector directly to the FPGA. Although the NetF-

PGA SUME does not allow to achieve a 100Gbps bandwidth through the available SFP+ interfaces, a typical 100Gbps application can achieve the required bandwidth by assembling a FPGA Mezzanine Card (FMC) daughter board providing additional network interfaces.

The memory subsystem comprises a DRAM memory, a flash memory, a microSD card slot and two SATA interfaces. NetFPGA SUME card is equipped with two on-board SODIMMS DRAM modules supporting up to 16GB. The board is supplied with two 4GB Double Data Rate Type Three (DDR3) modules officially supported by Xilinx Memory Interface Generator (MIG) cores. The on-board flash memory is intended to primarily store the FPGA's programming file. It can also store a initial bootup image loaded upon powering up. The microSD card provides a nonvolatile memory that can serve to supply a file system or as a location for debugging information.

NetFPGA SUME board is implemented as a dual-slot, full-size PCIe adapter which can be programmed both via the Joint Test Action Group (JTAG) connection and the PCIe interface. Moreover, it features a number of push-buttons and LEDs that can help in debugging the developed HW applications.

As a network prototyping device, the NetFPGA SUME board has been designed to operate in a variety of configurations. For instance, it can be used as a stand-alone unit outside of a PCIe host when equipped with a soft CPU IP core running a OS or a Firmware (FW). The computing system can be improved with a complete memory hierarchy by using the on-chip local memory as a cache for the CPU and by exploiting the on-board DDR3 modules. Moreover, up to two SATA HDDs can be used as storage devices. In the PCIe host interface configuration, the board acts as a network interface for a host PC and allows to prototype a 100Gbps physical network. Finally, as a 100Gbps switch, the board can be connected to other cards in order to implement a multi-FPGA computing system, which allows each node to achieve a overall 300Gbps bandwidth.

Although it is primarily intended for network application prototyping, NetFPGA SUME provides a multi-FPGA architecture which is similar to the one implemented in this thesis project. While both the solutions exploit PCIe inter-

face for host-board communication, NetFPGA achieves board-board transmission through a standard ethernet interface instead of a Aurora serial link.

MaxRing

Maxeler Architecture [14] works as a multi-FPGA system when it is composed by two or more DFE cards linked by a dedicated interconnection, called MaxRing. Moreover, a number of host systems comprising Maxeler DFEs can be connected by a standard network infrastructure. Maxeler Architecture will be discussed in detail in section 2.3.

2.3 Maxeler Architecture

Maxeler [14][57] is a family of FPGA-based HPC systems, provided by Maxeler Technologies, that exploit the dataflow model to accelerate the execution of complex computations. Currently, a variety of research centres and companies all over the world employ Maxeler systems in a broad range of applications, such as seismic datasets, financial risk and physics simulations.

Dataflow computers focus on optimizing the movement of data in an application and utilize massive parallelism between thousands of tiny dataflow cores. In the dataflow paradigm an application is considered as a dataflow graph of the executable actions, which is obtained from the source code. As soon as the operands for an action are valid, the action is executed and the result is forwarded to the next action in the graph. In most cases the constructed dataflow graph is static, therefore it can be compiled into a synchronous dataflow machine, suitable to be implemented in hardware. A node of the graph is a Multiplexer (MUX) or a buffer or an arithmetic operation, which takes a single clock cycle to be executed. In order to manage multiple data sets simultaneously, a dataflow machine can be replicated by exploiting the available HW resources.

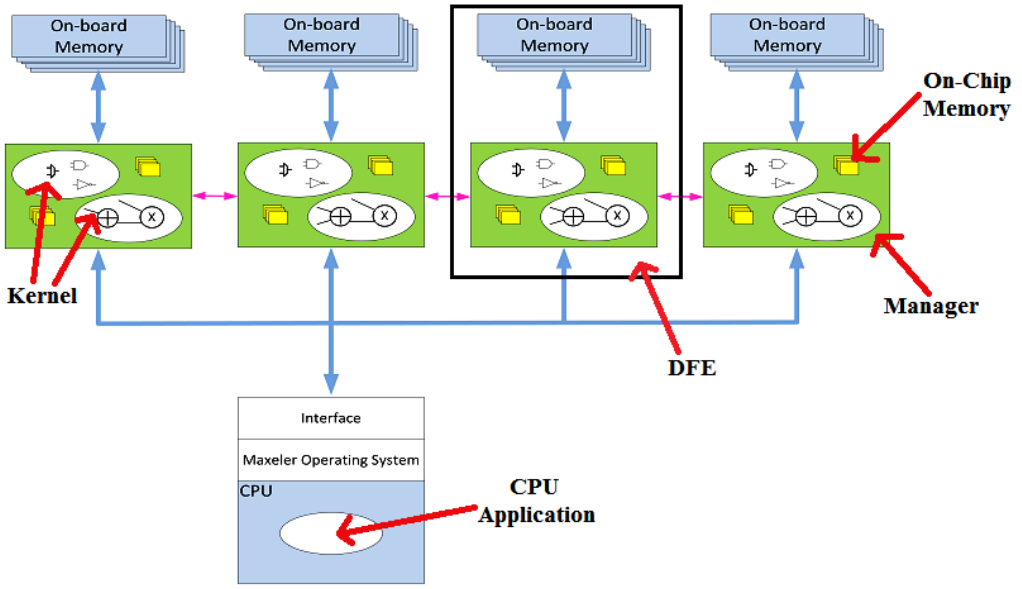


Figure 2.19: Maxeler Data Flow Architecture.

A Maxeler hardware accelerator system, as shown in figure 2.19, equips one or more FPGA boards, called DataFlow Engines (DFEs), connected to a host CPU via PCIe or Infiniband channels. Moreover, two or more DFEs can be linked through a high bandwidth board-board communication channel called MaxRing interconnection. Each FPGA accelerator can access two types of local memory: a on-chip memory which can store several megabits of data accessible with a high-bandwidth and a large on-board memory which can store many Gigabytes of data.

Maxeler provides several DFE architectures equipped with up to 48GB on-board RAM memory, featuring different FPGA devices from Xilinx and Altera. The architecture can be configured using the the tools developed by Maxeler, that comprise an Eclipse-based Integrated Development Environment (IDE) called MaxIDE, a custom compiler called MaxCompiler and MaxelerOS, an extension to Linux.

A general application to be accelerated in HW needs to be rewritten to fit the dataflow architecture. This process consists of writing three different program parts:

- the kernels, which implement the computational components of the appli-

cation in hardware;

- the manager configuration, which connects the kernels to CPUs, DFE memory, other kernels and other DFEs via the MaxRing interconnection;
- the software running on the CPU, where the DFEs are integrated into the original application using a simple Application Programming Interface (API) for transferring data to and from the DFEs directly and into and out of the RAM of the DFEs.

The software application for the CPU can be developed using C or Fortran language with the standard IDEs, while the kernels and the manager have to be written in MaxJ language, a Java-based programming language supported by MaxIDE. When all the parts are ready to be implemented, MaxCompiler merges the source code with the MaxelerOS APIs in order to generate the DFEs configuration files, that describe the operations, layout and connections of the dataflow engines. In a DFE configuration file a kernel is a streaming core with a dataflow described by a unidirectional graph, which comprises the arithmetic datapaths for the computations. A manager is implemented as a module orchestrating the data I/O for the kernels. The compiler inserts synchronization HW to ensure the correct timing for each DFE architecture, which features a manager and one or more kernels.

When the computation starts, data is streamed from the memory into the DFE chip where operations are performed and is forwarded directly from one computational unit to another, without being written to the off-chip memory until the chain of processing is complete. Once a program has finished running, the dataflow engine can be reconfigured for a new application in less than a second.

Maxeler provides a family of computing nodes suitable for data centers. A node can implement the Maxeler architecture as one of three different configurations: network sharing configuration, low latency configuration and high data transfer configuration.

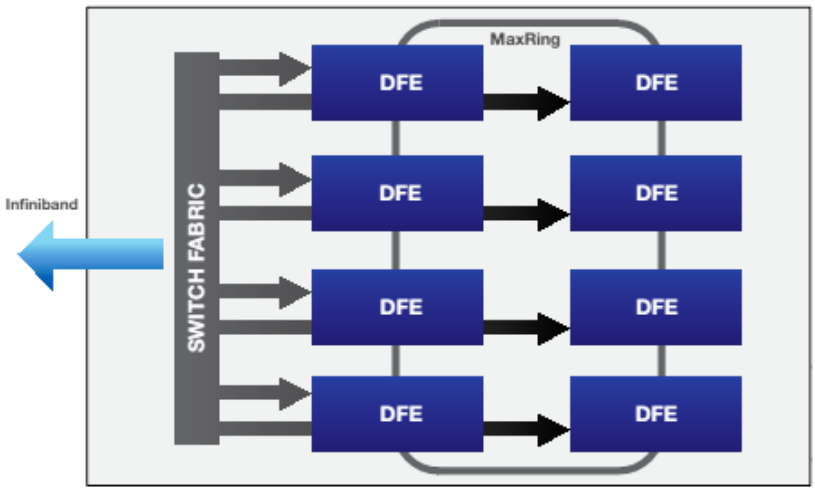


Figure 2.20: Network sharing configuration.

In Network sharing configuration, shown in figure 2.20, the Maxeler cluster node comprises multiple DFEs as shared resources on the network, allowing them to be used by the applications running anywhere in the cluster. The DFEs can be accessed by a remote CPU client machine via infiniband network, while multiple engines within the same node can communicate directly through the MaxRing interconnect. Remote CPU nodes can utilize as many DFEs as required for a particular application and release the DFEs for use by other nodes when not running computations. Client CPU nodes run standard Linux with Maxeler software, that automatically manages the resources within the nodes.

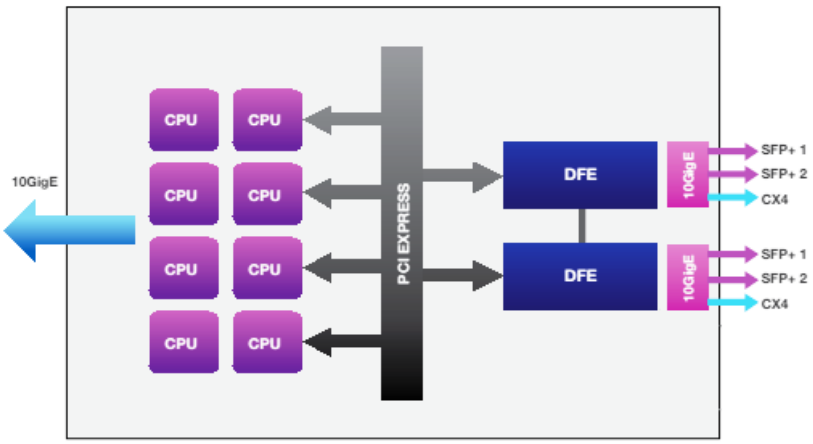


Figure 2.21: Low latency configuration.

Low latency configuration, depicted in figure 2.21, features a low latency network interface connected to both the CPUs and the DFEs on the same cluster node. This solution allows to accelerate the most timing-critical applications while exploiting two DFE units at the same time. Each cluster node can be connected to the infrastructure via ethernet link, while the DFEs can be directly connected to the other nodes using SFP transceivers. The CPUs on each node run a production-standard Linux distribution, with the DFE management software installed as a standard Linux package.

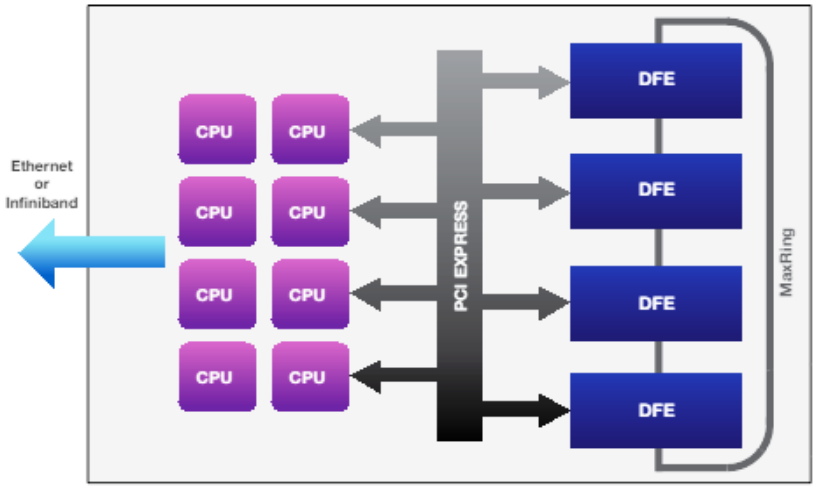


Figure 2.22: High data transfer configuration.

Figure 2.22 shows the high data transfer configuration, which couples the CPU units with up to four DFE engines connected through a MaxRing. This solution stands in the middle between the two preceding configurations. Indeed, the CPUs and the DFEs are included in the same cluster node as in the low latency configuration without sacrificing the high computing power provided by the network sharing configuration. However this solution is not suitable for extreme low latency requirements. Each node is connected to the infrastructure through infiniband or ethernet interconnection.

Although Maxeler offers a FPGA hardware platform coupled with a rich development environment, it requires the user to provide a complete description of the architecture that will be implemented in Hardware Description Language (HDL) code by the MaxCompiler tool. Moreover, the developer has to write the

hardware specification using MaxJ language, without having the option to use a more common language, such as C. Therefore, Maxeler requires the developer to have some knowledge in hardware design, without which the acceleration of a compute-intensive algorithm remains unachievable.

The solution provided by Maxeler is similar to the one proposed in this thesis project, as both are based on a multi-FPGA system composed of a host computer and a number of PCIe accelerators connected through a board-board high-performance link, each one hosting a chain of hardware accelerators.

3

Proposed Architecture

This chapter provides a description of the proposed design and a few qualitative power efficiency considerations. All the steps that led to the implementation of the architecture of a basic block are illustrated in section 3.1, by gradually presenting the achieved advancements, in order to highlight the components of the system and its features. The included subsections show how the final multi-FPGA system have been designed by adding new interfacing capabilities to the initial single-board accelerator. Section 3.2 illustrates the structure of a cluster node. The power efficiency considerations are presented in section 3.3.

3.1 The Basic Block

The Basic Block (BB) is the basic component of the *exaFPGA* [6] project. It is an FPGA-based hardware accelerator that stands at the lower level of the infrastructure. When two or more basic blocks are connected together and installed into a host computer, they constitute a Cluster Node (CN).

The following subsections explain how the basic block architecture has been designed with a step-by-step process, starting from a simple single-board accelerator.

3.1.1 The Starting Point: a Single-Board Accelerator

At the beginning of this thesis work the only available architecture for the *exFPGA* [6] project was the single-board hardware accelerator designed by *Natale and Sicignano* [7] to test the performance of their Streaming Stencil Time-steps (SSTs). The single-board accelerator had been developed as a *block design* with *Vivado design suite* [30] by exploiting its IP Integrator feature. The design was implemented and tested with a *Xilinx VC707* [29] board.

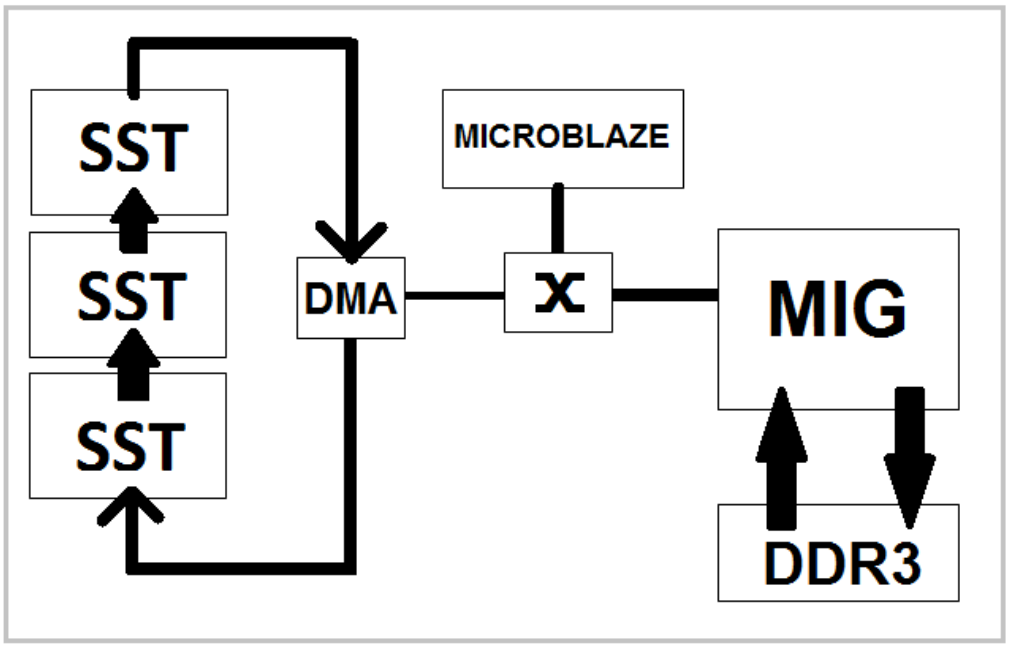


Figure 3.1: Single-board accelerator

As shown by figure 3.1, the accelerator is a complete computing system, including a Memory Interface Generator (MIG) [35] that allows the communication with the on-board DDR3 RAM memory and a DMA [34] that moves the data to and from the SSTs queue. All the computation is managed by a Firmware (FW) running on the Microblaze Central Processing Unit (CPU) [36] that instantiates the input data and controls the DMA [34] through its AXI4 lite interface. The data, loaded into the on board DDR3 memory, flow into the chain of SSTs through a DMA-managed AXI4 streaming bus. After being processed by the SSTs, they are stored again into the on-board memory.

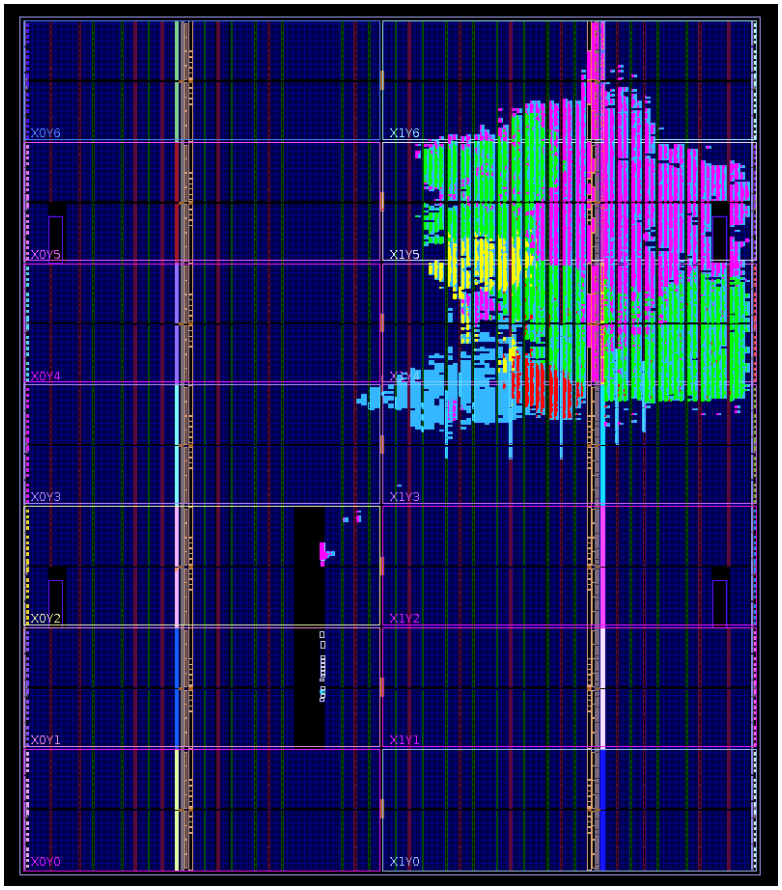


Figure 3.2: Virtex-7 floor plan of the Single-board accelerator

Figure 3.2 shows how the design has been implemented on a Virtex-7 FPGA. Most of the used area is occupied by the MIG [35], which is painted in magenta, and by the interconnection crossbar, which is painted in green. The DMA [34],which is highlighted in yellow, and the Microblaze CPU [36], which is highlighted in red, require only a fraction of the overall used resources.

This architecture is useful to test the SSTs, but it is clearly unfit for a HPC system aiming to achieve exascale performance, since it lacks the necessary interconnections to be scaled-up to a multi-FPGA system. Next sections show how high-performance board-board and host-board connections have been added to the architecture.

3.1.2 The first design: Aurora with AXI Chip2Chip

In order to build a working multi-FPGA system, a board-board interconnection is the obvious improvement to the previous architecture. Current FPGA boards feature a variety of protocols and interfaces that allow to connect a board to another.

Nowadays most HPC systems use *ethernet* or other custom technologies to connect their accelerator modules. Usually these interconnections are managed by an additional network infrastructure. This solution allows to implement a scalable infrastructure, by exploiting well-known reliable technologies. However these packet-based technologies are becoming more and more unfit for the modern HPC systems because they add an additional overhead to the payload. Moreover they need a special network infrastructure which reduces the performance and increases the cost of the system.

In this work a new solution for the board-board interconnection is presented. It exploits the high-speed serial link provided by *Xilinx* Field Programmable Gate Arrays (FPGAs) and uses coaxial cables to physically connect two or more boards.

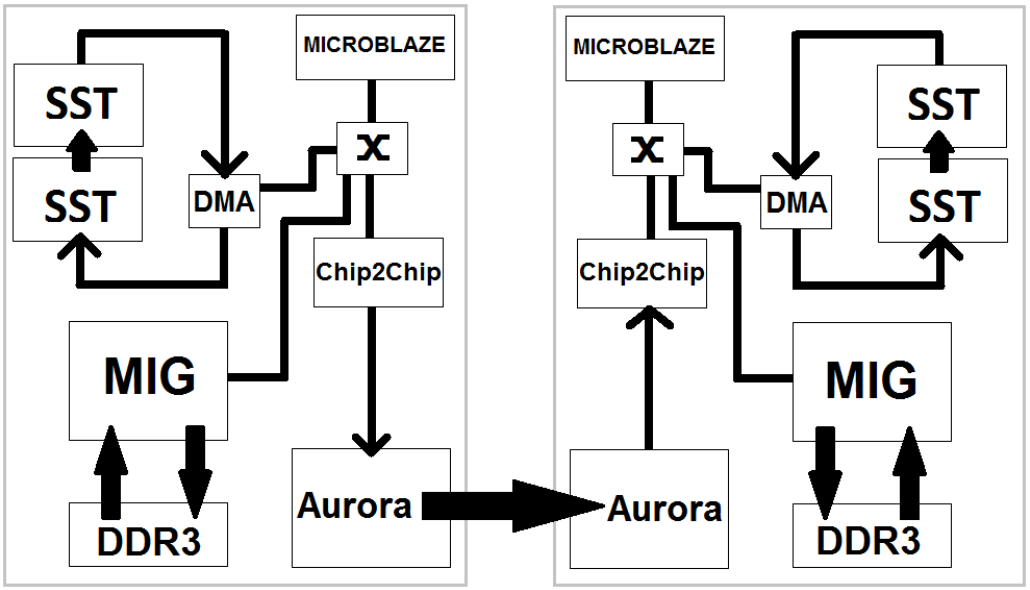


Figure 3.3: Board-board interconnection with Aurora and Chip2Chip

In figure 3.3 the first design is depicted. It consists of two *VC707* [29] boards

connected through a serial *Aurora* link.

The architecture is similar to the single-board accelerator explained above, except for two additional IP cores:

- *Xilinx Aurora 64B/66B* [32] is a free IP core included into *Vivado* [30] IP library. It exploits 7-series transceivers to provide a high data rate protocol for high-speed serial communication. The general purpose data channel achieves a throughput ranging from about 60MB/s to about 32GB/s, depending upon the number of lanes used and upon the clock frequency. This feature can be customized by changing the operating frequency of the GT transceivers through a suitable external clock generator and by increasing the number of lanes up to a maximum of 16. The core features an AXI4 streaming interface, that can be customized in order to work as a simplex/duplex framing or streaming interface. The data are transmitted with the 64B/66B encoding and the clock signal is embedded into the data stream. This IP core is suitable for board-board interconnection.

- *Xilinx Axi Chip2Chip* [33] is a free soft IP provided with *Vivado Design Suite* [30]. This core can be used to transparently bridge various AXI systems. It features an AXI Memory Map interface that can be customized in order to work either in master or in slave mode. Aurora IP can be easily managed by integrating it with an AXI Chip2Chip core.

The first design has been obtained by adding the serial link capability to the single-board accelerator. However, in order to implement a working board-board connection through Aurora, Axi Chip2Chip IP core has been used. Indeed, no technical paper providing useful hints for properly instantiating Aurora IP core without Axi Chip2Chip has been found.

Unfortunately Axi Chip2Chip exposes a single Axi Memory Map interface, therefore is can not work as a master and as a slave simultaneously. This feature forces the multi-board system to transmit the data in one direction only: from a board to the other and not the vice-versa. On the Transmitting (TX) board data are loaded into the DDR3 memory by a custom FW running on the Microblaze

CPU [36]. Then the on-board DMA [34] moves the data into the SSTs queue, where the computation takes place. At the end of the computation, the DMA [34] transfers the data to the slave Axi Chip2Chip IP core that interfaces with Aurora core and exploits its high-speed link to move the data to the second board. On the Receiving (RX) side, after the data have reached the master Axi Chip2Chip IP core through Aurora, they are stored into the on-board DDR3 memory. This is a mandatory step, because the DMA IP core [34] provided by *Vivado* IP library exposes a master Axi memory map interface, therefore it can not be mapped into the memory space of the Axi Chip2Chip that features a master interface too. The FW running on the on-board Microblaze CPU [36] is kept into a wait state until all the data have been stored into the memory. Then it starts a transfer from the DDR3 to the SSTs queue by managing the DMA [34]. After the data have been processed, they are stored again into the on-board memory.

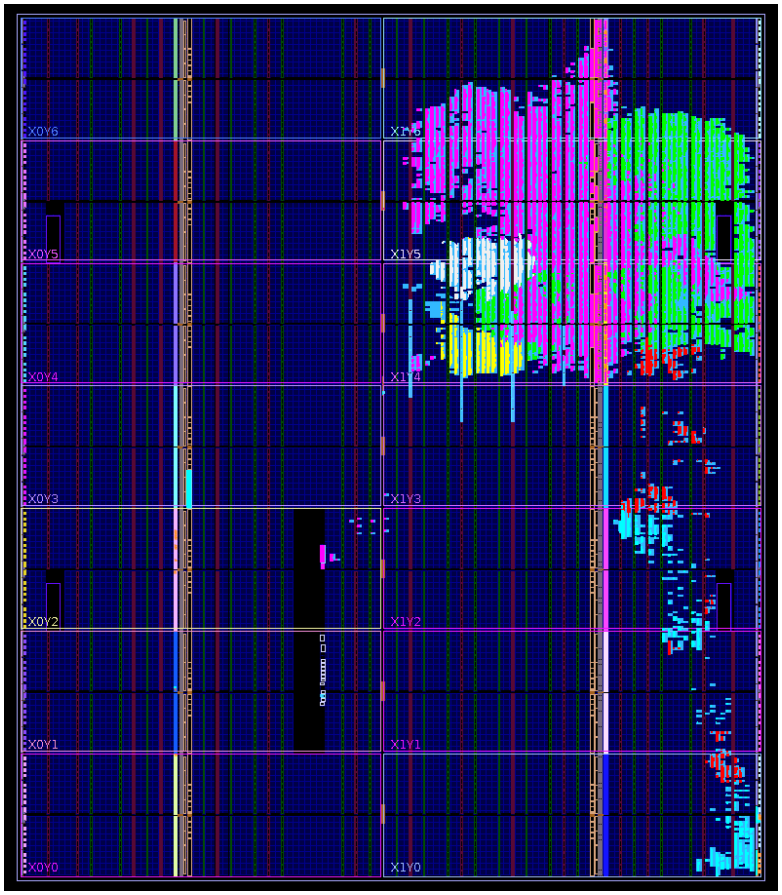


Figure 3.4: Virtex-7 floor plan of one of the two boards of the first design

Figure 3.4 shows the floor plan of one of the two FPGAs composing the implemented design. As in the previous section, the MIG [35], which is painted in magenta, is the IP that requires the largest amount of area. Also the interconnection crossbar, which is painted in green, is a resource-consuming core. The Microblaze CPU [36] is painted in yellow and the DMA [34] is painted in white. Aurora IP core, which is painted in blue, requires only a fraction of the total used resources. Axi Chip2Chip, which is painted in red, is similar to Aurora with respect to the area utilization.

The proposed solution allows to easily implement a multi-board system through a high-speed serial link, unfortunately it has various drawbacks. First, it requires a complete computing system to be implemented on both boards, this requires a number of Intellectual Properties (IPs), thus increasing the resource utilization. Moreover the system is managed by the two special FWs running on the on-board Microblaze CPUs [36], therefore the overall performances are seriously reduced. Another drawback is due to the presence of the Axi Chip2Chip IP core: it forces the data to be stored two times on the RX board because it can not be directly linked to the DMA [34], moreover it integrates the Aurora core by exposing a memory map interface instead of the streaming interface provided by Aurora. These features cause serious limitations to the performance of the system.

3.1.3 The second design: A fully-streaming Serial Link

As clearly explained in the previous subsection, various limitations made the first design unfit for a scalable HPC system. Therefore a new architecture has been designed, featuring a fully-streaming board-board link.

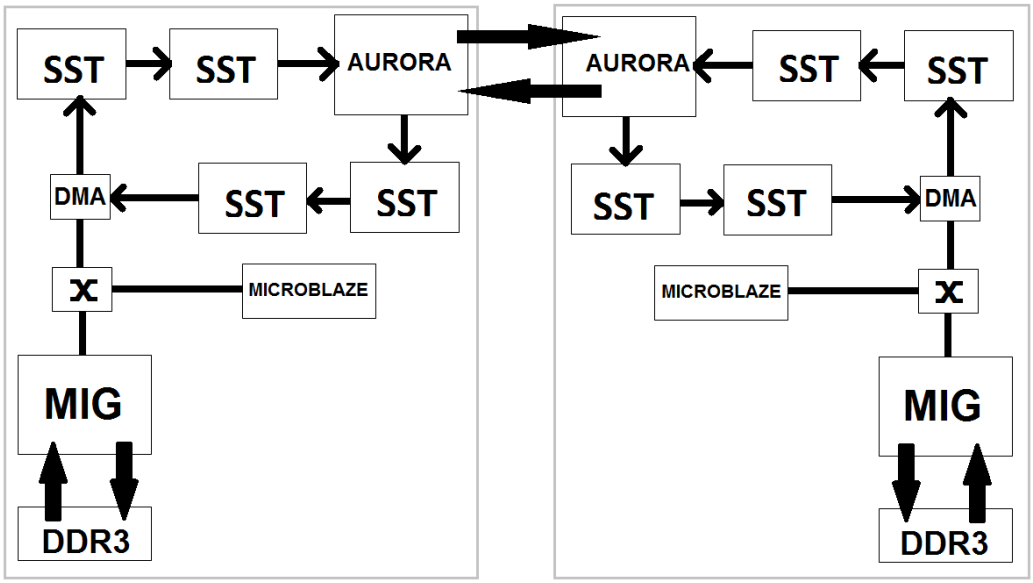


Figure 3.5: Board-board interconnection with Aurora streaming

The second design is illustrated by figure 3.5. The architecture derived from the first design has been modified by connecting the Aurora IP core to the streaming interface of the DMA [34], without the Axi Chip2Chip.

This considerable improvement has been achieved by carefully analysing the signal waveforms included into the *Aurora Product Guide* [66] in order to understand the initialization of the link and the management of the clock and reset signals. Therefore, Axi Chip2Chip IP core has been replaced by two custom cores that allow to properly manage the Aurora IP core without overriding its AXI4 streaming interface.

The first custom core is a *Verilog* module designed with *Vivado* [30]. It includes a Finite State Machine (FSM) that takes care of the initialization phase of the Aurora IP core. Moreover it manages the recovery phase when the serial link undergoes a failure. The second custom core has been implemented with *Vivado HLS* [31], starting from its *C-language* specification. It is in charge to generate a reset signal for the IPs connected to the streaming interface of the Aurora IP core. Indeed, Aurora IP provides a clock signal associated to its streaming interface, however it lacks of a reset signal.

Furthermore, two asynchronous First In First Out (FIFO) IP cores must be

placed before the AXI4 streaming interfaces of each Aurora IP core in order to achieve the timing closure of the entire architecture. This step is mandatory, because two different clock domains coexist in the same circuit: the first is provided by Aurora IP core through its streaming interfaces, the second is enforced by the MIG IP core [35], that is in charge to generate the clock signal for the system. By adding a asynchronous FIFO at the boundary of the two clock domains, the timing closure is met. Indeed, first the stream of data enters the FIFO at a rate provided by the input clock signal, then it exits the FIFO synchronously with the output clock signal, this provides a clear separation between the two clock domains.

As in the previous design, the data are loaded into the DDR3 memory of the first board of the system. Then they are moved to the streaming SSTs queue by the DMA [34]. After the computation, the data reach Aurora IP core, that transmit them to the other board where they are processed by another SSTs queue. Finally, they are stored into the DDR3 memory of the second board. Thanks to the full-duplex interconnections, the data can be moved from the second to the first board by traversing two additional SSTs queues.

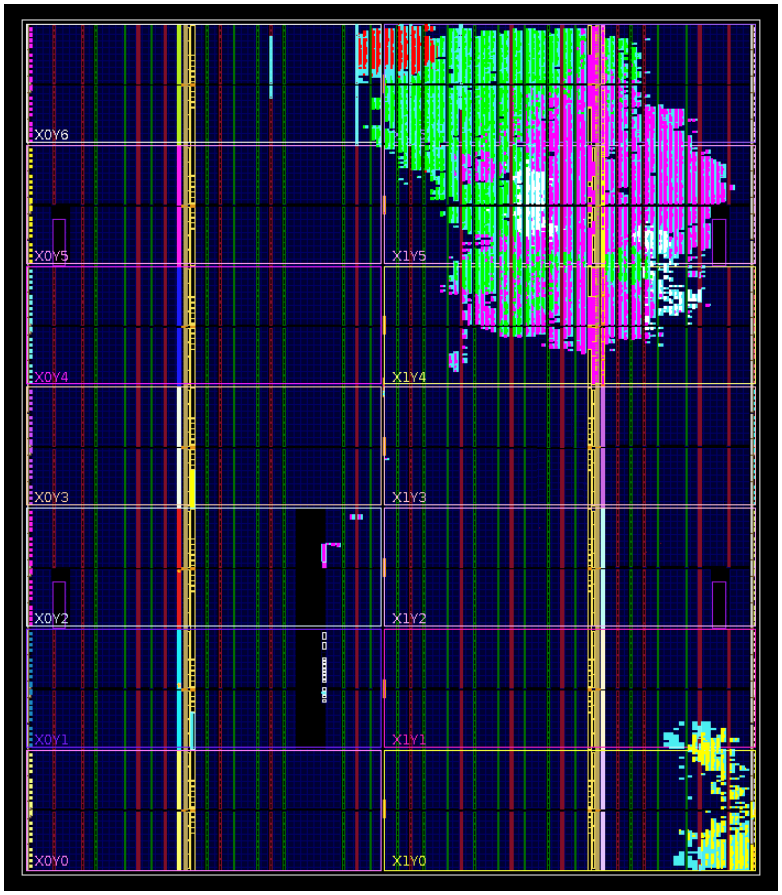


Figure 3.6: Virtex-7 floor plan of one of the two boards of the second design

The Virtex-7 floor plan of the second architecture is depicted in figure 3.6. The Microblaze CPU [36] is painted in red, the DMA [34] is painted in white. Most of the used area is occupied by the interconnection crossbar, which is painted in green, and by the MIG [35], which is painted in magenta. Aurora IP core is the yellow area shown in the bottom-right corner of the picture. Notice that the area occupied by the Axi Chip2Chip in the previous design is now free.

This design allows to implement a basic multi-FPGA system, moreover the number of interconnected boards can be increased by adding more accelerators. Therefore this is a scalable solution.

An important capability is the fully-streamed bus that allows the data to be moved without any bottleneck through the SSTs queue. This permits to achieve a high throughput during the processing phase and it also simplifies the architecture. Moreover the link can be crossed in both directions, thus increasing the

flexibility of the system.

The implemented design increases also the efficiency of the architecture by reducing the number of accesses to the on-board memory. Indeed, the data are moved directly from a board to another without any intermediate access to the DDR3 memory. However the proposed design still needs to be managed by a FW running on a Microblaze CPU [36], therefore the resource utilization increases and the overall performances do not increase as well.

Finally the system is unable to interface with a host PC through a high speed link, hence the data need to be pre-loaded into the on-board DDR3 memory before the beginning of the computation, thus breaking down the overall performances. Next section shows how a PCIe interface has been added to the implemented architecture.

3.1.4 The third design: PCIe host-board Connection

The architecture illustrated in the preceding subsection can be scaled-up to a complex multi-board system, however it lacks a high-speed link to interface with a host system. This section shows how a PCIe interface has been developed, by exploiting a commercial IP core.

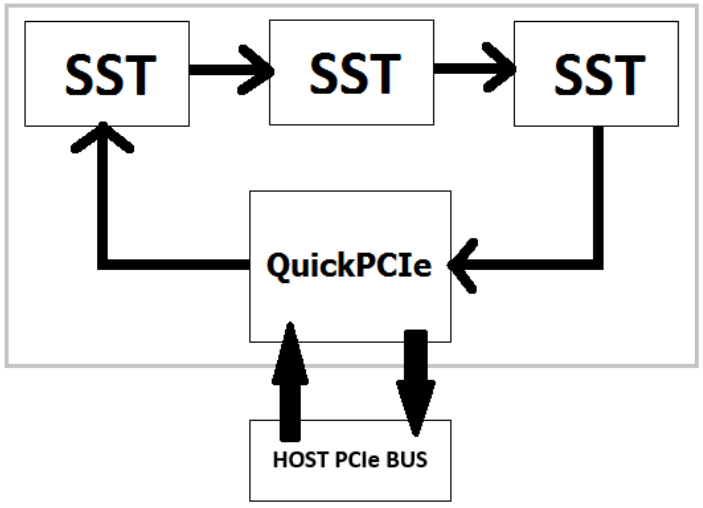


Figure 3.7: Host-board PCIe interconnection

Figure 3.7 illustrates the implemented architecture. It is a self loop composed

of a PCIe interface IP core and of a queue of SSTs, connected on a AXI4 streaming bus. *QuickPCIe* [25], the core used in this design, is a commercial IP provided by *PLDA* [22], an international company that delivers various PCIe solutions. This soft IP integrates Xilinx hard PCIe IP core and provides DMA capability and AXI4 streaming compliance.

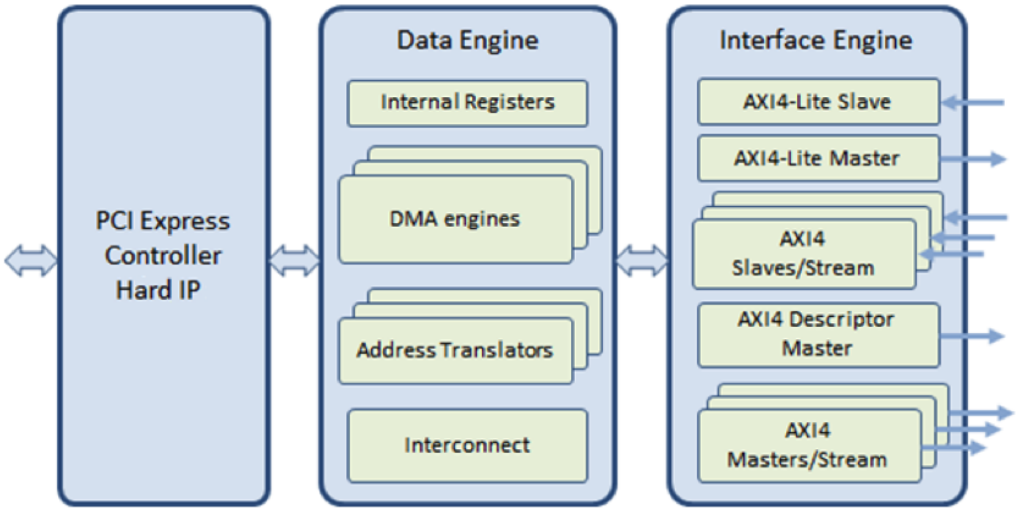


Figure 3.8: QuickPCIe Block Diagram

As depicted by figure 3.8, the core is composed of three layers.

- The lower level includes the PCIe hard IP provided by Xilinx. It implements the basic features of the protocol and exposes a suitable interface to the upper layer.
- The higher layer implements the AXI capability. It provides various AXI4 interfaces to the custom logic connected to QuickPCIe IP core.
- The layer in the middle acts as a bridge between the other two levels. Therefore it interconnects the hard PCIe IP with the AXI interfaces. Moreover it includes a number of DMA engines, thus adding the DMA capability to the basic PCIe interface. Finally, it provides the Clock Domain Crossing (CDC) capability, that allows to safely instantiate QuickPCIe IP core into a design with multiple clock domains.

QuickPCIe is compliant with the PCIe Eight-Lane (x8) second generation in-

terface and it can be customized in order to exploit the GTX transceivers of a Virtex-7 FPGA. Before QuickPCIe has been chosen for this work, several free PCIe IP cores had been tested:

- *Xillybus* [37], a commercial IP developed by the company of the same name;
- *Riffa 2.0* [26], provided by the University of California, San Diego;
- *EPEE Library* [5], designed by the Chinese Center for Energy-efficient Computing and applications;
- *AXI Bridge for PCI Express (PCIe) Gen3 Subsystem* [2] and *7 Series Gen2 Integrated Block for PCI Express (PCIe)* [1], provided by Xilinx through the Vivado IP library.

These IP cores implement the PCIe interface by providing different solutions, some of which are very interesting. However, they were discarded because in the most cases they lack some basic feature *i.e.* they provide custom interfaces instead of the AXI4 ones, they do not have a working driver, they do not support the architectures with multiple clock domains. Furthermore, some of them do not work as expected and some others do not work at all. In particular, the two IP cores provided by Xilinx are not suitable to be used alone, because they need to be integrated by additional custom logic. Moreover a clear explanation on how to instantiate them into a Vivado block design has not been found.

Unlike the other IPs, QuickPCIe includes a working reference design, a working demo application, a clear documentation, a driver for a variety of Operating Systems (OSs) and a easy-to-use set of Application Programming Interfaces (APIs). Unfortunately, QuickPCIe is not ready to be instantiated into a Vivado block design, therefore it needs to be customized in order to be wrapped into a Vivado IP core.

In order to achieve a working PCIe IP core, the reference design provided with QuickPCIe has been chosen as a starting point for additional improvements, because it has proven to be already well-integrated with the software driver.

Therefore, all the Verilog modules composing the reference design have been imported into a new Vivado project and the correct hierarchy has been rebuilt.

Then two important changes to the Verilog specification have been made:

- two AXI4 streaming interfaces have been added to the wrapper of the core. They act as exeternal input/output interfaces when the core is instantiated into a Vivado block design;
- additional logic has been included into the interface layer, it is in charge to manage the new AXI4 streaming interfaces.

Then the project has been wrapped and a Vivado IP core has been generated. Finally, the new IP core has been instantiated into a blank Vivado IP integrator project and a queue of SSTs has been connected as a self-loop to the AXI4 streaming interfaces.

The design is managed by a custom application running on the host system. It has been developed from scratch by exploiting the APIs provided with QuickPCIe IP core. First, the application initializes the core and the driver, then it instantiates a buffer on the host RAM memory and fills it with the input data for the SSTs queue. After that, it starts a DMA transfer from the buffer to the AXI4 streaming interface of the PCIe IP core. At the end of the computation the application starts a DMA transfer from the board to the host system and stores the processed data into the RAM memory of the PC.

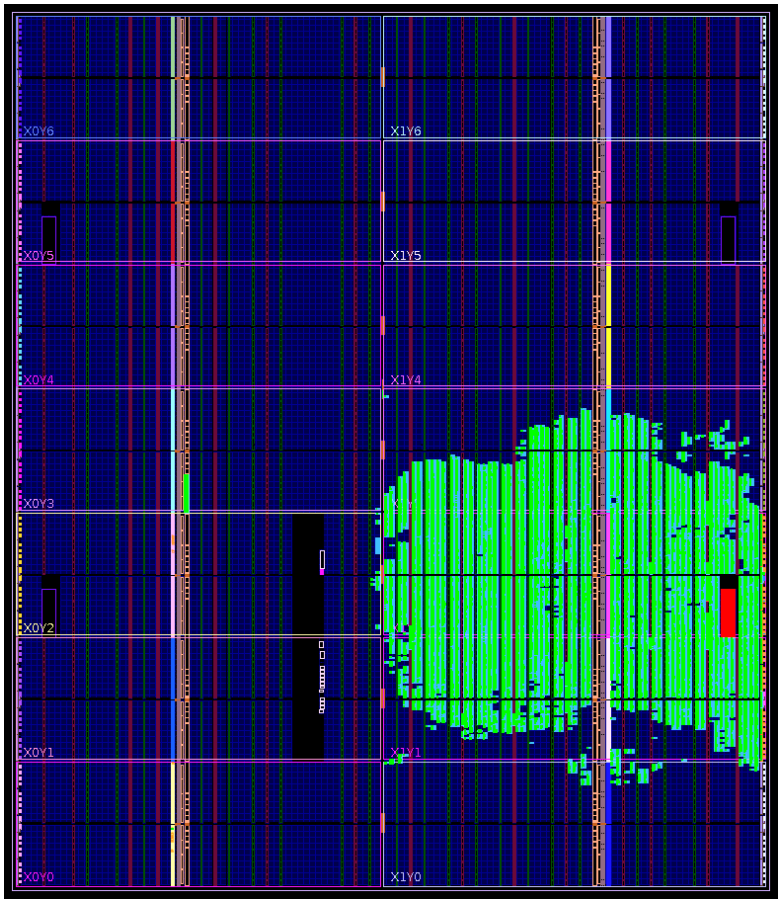


Figure 3.9: Virtex-7 floor plan of the design with QuickPCIe only

Figure 3.9 shows the floor plan of the third design. The red-painted area is occupied by the Xilinx PCIe hard IP core. QuickPCIe IP core, which is painted in green, requires a considerable amount of resources to be implemented. However the reference design provided with the evaluation version, which has been used for this thesis project, includes a number of demo features. Therefore, when using the purchased version of the IP in spite of the evaluation one, only the necessary features are included into the core, hence the required amount of area is significantly reduced.

The implemented design allows to interface a single-board accelerator with a host PC. This architecture features a fully-streaming bus and does not require any custom FW to be developed. Moreover it achieves a good resource utilization, because the design requires no CPU, no DMA [34], no MIG [35] and, obviously, no crossbar.

3.1.5 The final version: A multi-board Pipeline

This section describes the final version of the proposed design. As explained in the previous sections, it is the result of an incremental process, that has led to the second and to the third design.

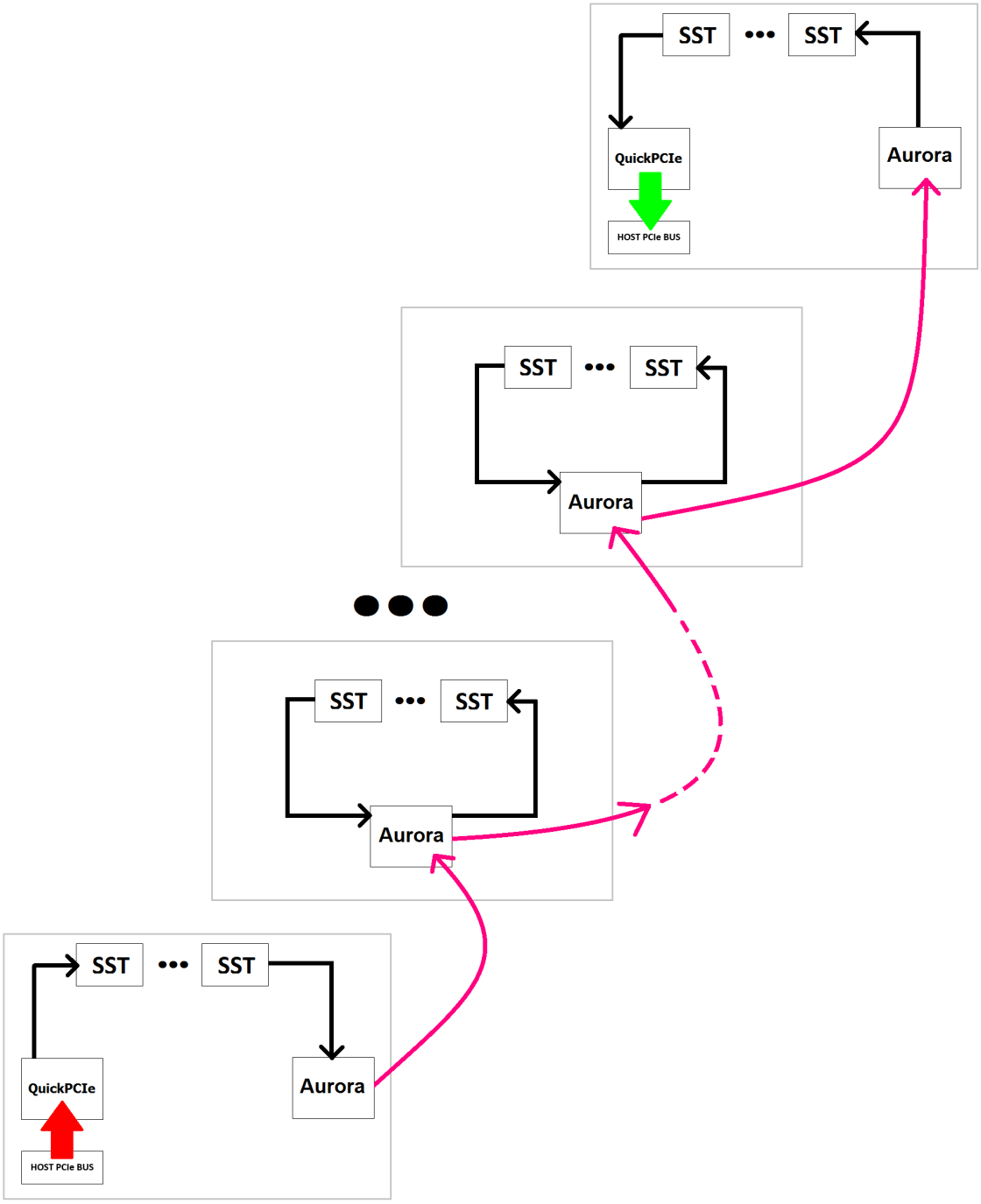


Figure 3.10: The multi-board Pipeline

Figure 3.10 illustrates the final architecture. It has been implemented by merging the second and the third design, in order to exploit their most important features: the host-board interconnection and the board-board link.

The overall system is composed of a queue of FPGA boards, interconnected with coaxial cables. Notice that only the first board and the last board include a PCIe interface. The other boards feature only a serial link, because the exchange of data with the host system is performed only at the ends of the queue. The only resources needed by the proposed design are those required by the communication interfaces, the remaining resources are available for the computing logic. Indeed, no CPU is implemented into the design and no MIG [35] is needed. Furthermore no FW needs to be developed. A SSTs queue is implemented on each FPGA board.

The system is managed by a custom application running on the host PC. First, the data are initialized on the host PC, then they are moved to the first board of the queue by exploiting the PCIe x8 interface (red arrow) where they are processed by the SSTs queue. At the end of the computation, the data reach the Aurora IP core, that moves them to the second board of the system. The data traverse all the boards composing the queue through the coaxial cables (magenta arrows). Each time the stream of data reaches a board, it passes through the SSTs queue connected to the AXI4 streaming interfaces of the Aurora IP core. Then the computed data are moved to the next board. Finally, the data are moved to the last board, where they pass through the on-board SSTs queue and reach the PCIe IP core, that transfers them to the RAM memory of the host PC (green arrow).

Notice that no asynchronous FIFO is required. Indeed, the CDC feature of QuickPCIe IP core is enough for the design to achieve the timing-closure, because it splits the clock domain of the host PCIe infrastructure and the clock domain provided by Aurora.

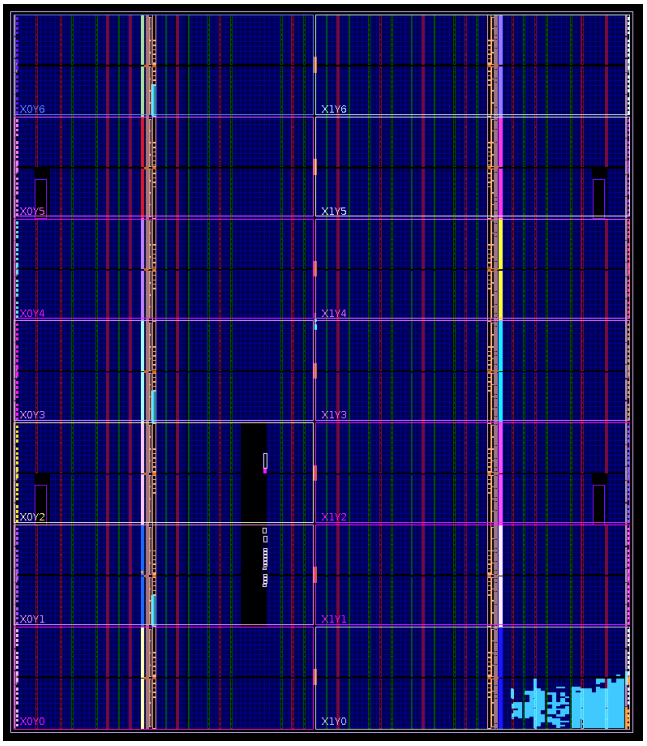


Figure 3.11: Virtex-7 floor plan of the design including Aurora only

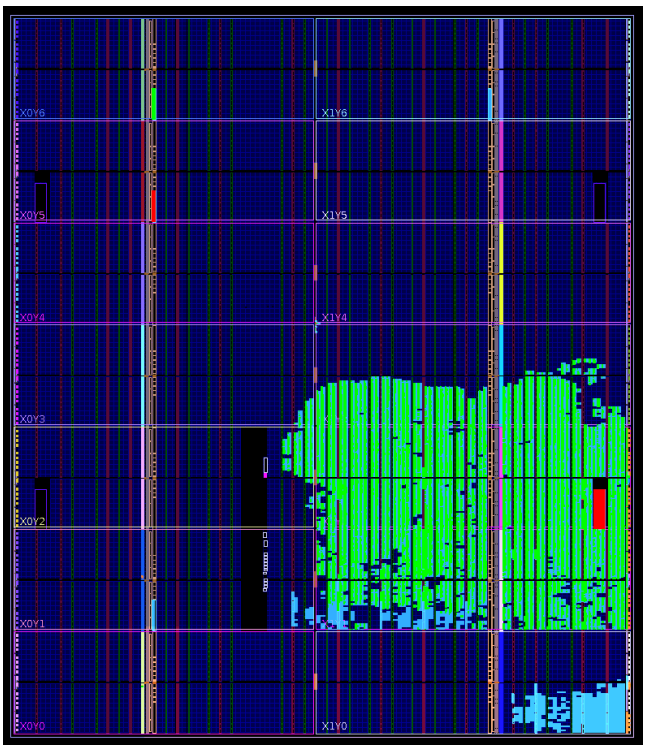


Figure 3.12: Virtex-7 floor plan of the design including QuickPCIe and Aurora

Figure 3.11 shows the floor plan of one of the middle boards of the system. The only implemented core is Aurora, located into the bottom-right side of the picture. As can be clearly noticed, the great majority of the area of the FPGA is free, hence it is available for the SSTs.

Figure 3.12 illustrates the floor plan of one of the two FPGAs connected with the host PC. Nearly all the used area is occupied by QuickPCIe, which is painted in green, and by Aurora, which is painted in blue. However, most of the area is free and can be used for the computing logic.

The proposed architecture has various capabilities:

- It can be scaled both by increasing the length of the on-board SSTs queues, both by adding more boards to the system. This is an important feature with respect to HPC field for which the proposed design has been developed;
- The resource utilization is efficient and most of the area on each FPGA is available for the SSTs, hence it is reserved for computation;
- The power efficiency increases with the number of the boards included into the system, because the most demanding components are the PCIe interfaces at the ends of the queue;
- The achievable bandwidth varies upon the used FPGA boards, upon the clock source and upon the width of the interconnection links.The PCIe interface can reach a theoretical bandwidth of 7.88 GB/s if a generation 3 x8 PCIe IP core is used. Moreover the Aurora serial link can be operated with at a 32 GB/s theoretical bandwidth when in 16-lanes configuration and GTH transceivers are used.
- The frequency at which the SSTs queue processes the data stream is limited by the maximum frequency at which the AXI4 streaming bus can operate. This, in turn, depends upon the configuration of the two interface IP cores.
- The overall design is a pipeline composed of pipelines, therefore it can process a continuous stream of data without any interruption, thus providing a high-throughput HPC system.

The proposed architecture meets all the features required by a supercomputing system, therefore it can be added to a High Performance Computing (HPC) cluster node.

3.2 The Cluster Node

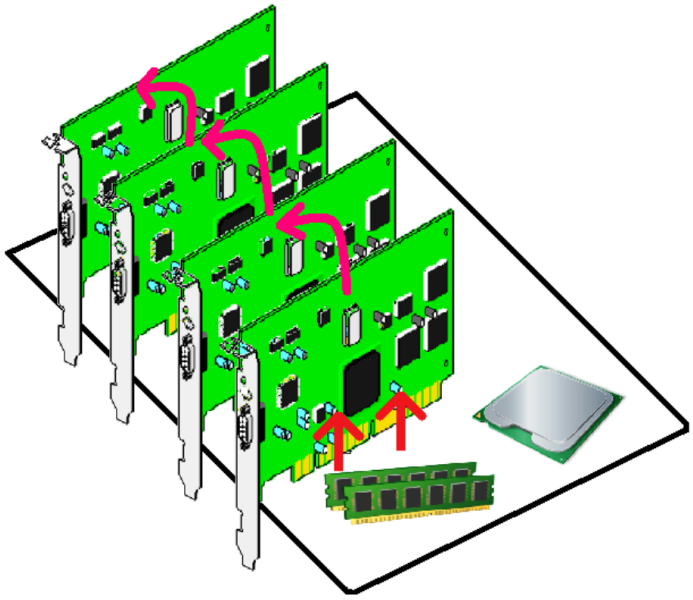


Figure 3.13: A Cluster Node

Figure 3.13 depicts the basic structure of a cluster node. It is composed of a host PC providing a server-class CPU and several PCIe slots. The architecture described in the previous section is integrated with the host system by connecting the FPGA boards into the PCIe slots. Notice that, even if all the boards are connected into the PCIe slots, only the two at the ends feature a PCIe interface.

The cluster node is managed by a special application running on its CPU, the input data and the computed data are stored into the local DDR3 memory. The DMA of the host system is exploited to move the data from the local DDR3 memory to the queue of FPGA boards and vice versa.

The cluster node can be connected to other cluster nodes to implement a more

complex HPC system. This can be achieved by connecting each cluster node to a suitable interconnection network. However, this goes beyond the purpose of this thesis project.

3.3 Power Efficiency Considerations

This section presents some considerations on the power efficiency of a generic multi-FPGA architecture when increasing the number of used FPGA devices and the number of accelerators instantiated on each device. These considerations are useful to understand, from a qualitative point of view, the power efficiency trend of the presented architecture.

A more detailed quantitative analysis of the power efficiency trend of a multi-FPGA system is provided in chapter 4.

3.3.1 Hypotheses

Before illustrating the considered architecture, it is necessary to explain the hypotheses that have been made and the base components of the architecture itself.

- The first hypothesis states that there is a direct proportionality between the number of executed computations and the amount of resources used for each configuration of the system. Indeed, in a multi-FPGA architecture the computations are performed by the hardware accelerators instantiated on each FPGA device, therefore, to perform more computations, more resources must be used to host a greater number of accelerators.

- The second hypothesis is similar to the first one as it states that there is a direct proportionality between the amount of dissipated dynamic power and the number of executed computations for each configuration of the system. Indeed, in order to increase the number of executed computations, a greater number of accelerators needs to be instantiated, thus increasing the power required to perform the computations.

- Each computing device that constitutes the system features a FPGA that allows to instantiate a large amount of computing resources, in the form of hardware accelerators. Moreover, each device can be equipped with a PCIe interface and with up-to two Aurora interfaces. In the first part of the presentation it is assumed that the power consumption of the Aurora device-to-device link is much greater than the power consumption of the PCIe interface ($Pw_{AUR} >> Pw_{PCIe}$). Therefore, the power consumption of the PCIe interface is negligible. This assumption will be discussed later.

Figure 3.14 shows a computing device that features both the Peripheral Component Interconnect Express (PCIe) and the Aurora interface, while figure 3.15 presents a device equipped with two Aurora interfaces: one for the input link and the other for the output link. These are the basic components of the considered system.

Notice that all the logic that is available on both devices can be exploited to perform the computations.

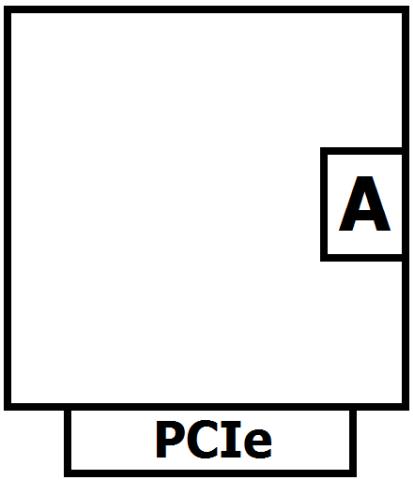


Figure 3.14: A computing device featuring both a PCIe interface and a Aurora interface.

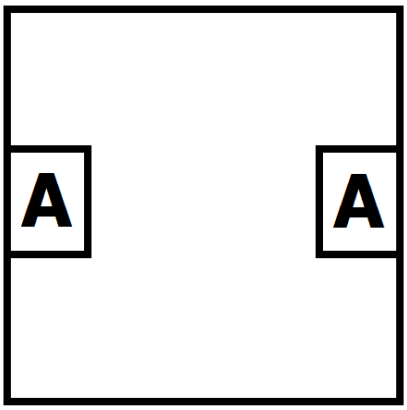


Figure 3.15: A computing device featuring a Aurora input interface and a Aurora output interface.

The assumptions that have been made previously are broad and they do not depend on numerical parameters. Although they are not valid for every type of workload, they are valid for the workloads used for this thesis project.

3.3.2 Building the System

In order to understand the overall power efficiency trend of a multi-FPGA system it is necessary to focus on a few simple configurations of the system itself. Therefore, three configurations have been analysed: a single computing device, a couple of devices linked together and a complete architecture, composed of a number of interconnected devices.

The first configuration, depicted in figure 3.16, comprises only a computing device, which features a PCIe interface. All the available resources can be used to instantiate the hardware accelerators. Notice that a Aurora link is not required, since the system does not need to interface with other computing devices.

In this case the main source of power consumption is the computing logic, which executes all the computations provided by the system.

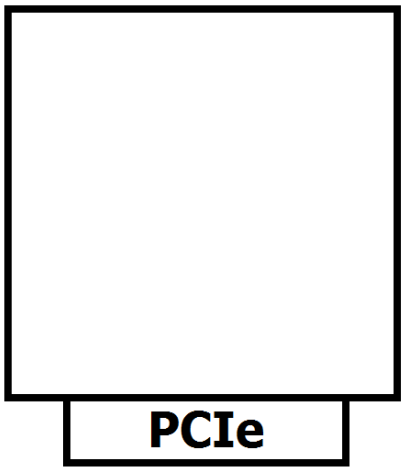


Figure 3.16: The first configuration: a single computing device featuring a PCIe interface.

The second configuration, shown in figure 3.17, is composed of two computing devices, both featuring a PCIe interface. The two devices are connected through a Aurora link, therefore a Aurora interface must be instantiated on each device, on the first as an output interface, on the second as an input interface. The available logic is slightly less with respect to the single device case, since a small amount of resources on each device are used by the Aurora interface.

Also in this case the main source of power consumption is the computing logic. However, a fraction of the total dissipated power is consumed to transfer the data through the Aurora link.

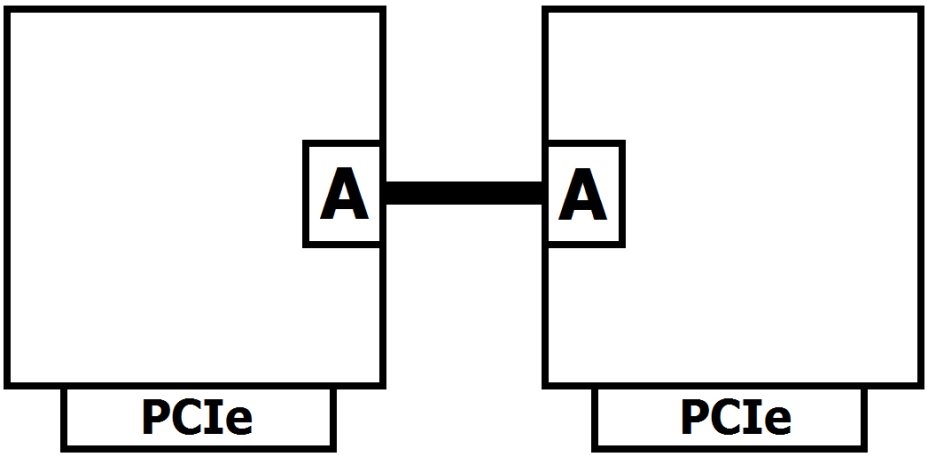


Figure 3.17: The second configuration: two computing devices featuring a PCIe interface and connected through a Aurora link.

The third configuration, shown in figure 3.18, comprises a variable number of computing devices, connected together as a queue. The first and the last device in the queue feature a PCIe interface and a Aurora link. The other devices are equipped with two Aurora interfaces, one for the input data and the other for the output data. They do not need a PCIe interface.

The length of the queue can be increased by adding more intermediate devices, each one featuring two Aurora interfaces.

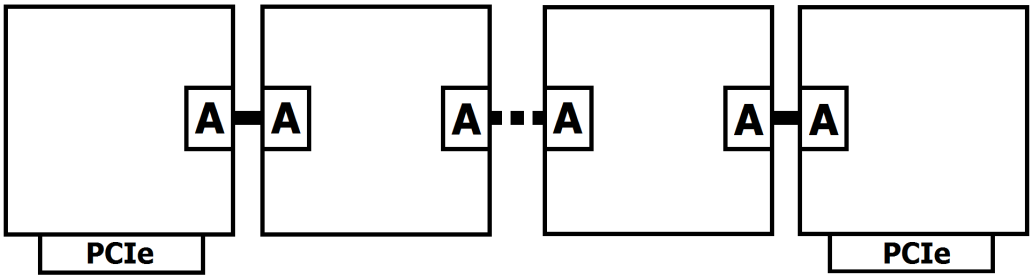


Figure 3.18: The third configuration: a queue of computing devices connected through a Aurora link. The two devices at the extremes feature also a PCIe interface

3.3.3 Power Efficiency Trend

The power efficiency trend of the considered multi-FPGA system follows the general equation of the power efficiency:

$$\mathrm{PowerEfficiency} = \frac{\mathrm{Throughput}}{\mathrm{PowerConsumption}} = \frac{\frac{\mathrm{PerformedComputations}}{\mathrm{ExecutionTime}}}{\mathrm{PowerConsumption}} \quad (3.1)$$

The equation 3.1 states that the power efficiency of a computing system is determined both by the number of computations performed during the execution time and by the power consumption of the system. When considering the presented multi-FPGA architecture, the number of performed computations is provided by the accelerators instantiated on the computing devices and the power is consumed mainly by the accelerators and by the inter-device link.

Therefore, the three configurations can be individually examined.

- The first configuration is also the most power efficient, since all the power dissipated by the device is spent to perform the required computations. Starting from a single device with no accelerators and increasing the number of accelerators, the power efficiency of the device quickly grows up-to a limit, which is also the upper-bound for the whole system.

- The second configuration is obtained by coupling the device used in the first configuration with another identical device. Therefore, a device-to-device connection is required. At the beginning the efficiency of the system is significantly reduced, since the newly added device hosts no accelerator. The overall power efficiency grows while the number of accelerators instantiated on the second device increases, until no more accelerator can be hosted by the two devices. At this point the second configuration provides the double of executed computations with respect to the first configuration. However it requires also the double of power, with the addition of a significant contribution brought by the Aurora link. Indeed, the two interfaces require some power to the devices in order to move the data through the link. Therefore, the power efficiency of the second configuration is less than the power efficiency of the first configuration.

- The third configuration is based on the second configuration with the addition of a variable number of intermediate devices. As in the previous case, the newly added intermediate devices host no accelerators, therefore the overall power efficiency is reduced. However, while increasing the number of hosted accelerators, the overall power efficiency tends to limit value, which is the power efficiency of a single intermediate device featuring two Aurora interfaces, and the contribution of the two devices at the extremes becomes more and more negligible. The limit value reached by this configuration is slightly less than the one achieved by the second configuration, since the intermediate boards, that feature two Aurora interfaces, consume more power than the two boards at the extremes, that are equipped with a

single Aurora interface.

3.3.4 A different Assumption

At the beginning of this section, it has been assumed that the power consumption of the Aurora device-to-device link was much greater than the power consumption of the PCIe interface ($Pw_{AUR} >> Pw_{PCIe}$). Therefore, the power consumption of the PCIe interface has been neglected.

From here on, the opposite assumption is taken, in order to analyse the power efficiency trend of the same multi-FPGA system when the power consumption of the PCIe interface is much greater than the power consumption of the Aurora device-to-device link ($Pw_{PCIe} >> Pw_{AUR}$). All the other hypotheses made previously are still valid.

The power efficiency of a single computing device follows the same trend shown under the previous assumption. However, in this case the maximum value reached by the single-device configuration does not set a upper bound to the entire system. After the local maximum has been reached, the power efficiency decreases due to the addition of the second device. The maximum power efficiency level that can be achieved with a two-device configuration is slightly less than the maximum value reached by the single-device configuration. The third portion of the curve is different with respect to the previous case, since the power efficiency of the complete multi-FPGA architecture is improved by the addition of the intermediate devices, that provide a greater power efficiency with respect to the single device equipped with the PCIe interface. As the number of hosted accelerators increases, the power efficiency tends to a upper bound, which is the power efficiency of a single intermediate device featuring two Aurora interfaces. Indeed, when considering long queues of devices, the power efficiency of the system is bounded only by the power efficiency of the single intermediate devices, since the contribution of the PCIe interface of the two devices at the extremes is negligible.

4

Experimental Evaluations

In this chapter the experimental evaluations for the proposed architecture are presented. The results are described in section 4.3, preceded by the experimental setup in section 4.1 and by the used benchmarks in section 4.2. Finally, section 4.4 presents a quantitative model for the power efficiency and its validation against the experimental results.

4.1 Experimental Setup

The proposed architecture has been tested with *Vivado Design Suite* [30]. *Vivado HLS* (v2015.2) [31] has been used to generate the Register-Transfer Level (RTL) and the Vivado's Intellectual Property (IP) core for each Streaming Stencil Time-step (SST) module, starting from its C language specification.

Both the generation of the bitstream and the programming phase have been performed using *Vivado* (v2015.2) [30]. Synthesis and implementation have been executed with an AMD Athlon II X4 640, featuring an 8GB DDR3 RAM. Due to the strict requirements for synthesis and implementation, less than half of the available resources on the FPGA could be exploited.

All the tests have been performed with two *VC707*[29] boards and with a host PC running Ubuntu 14.04 x64, the same used for synthesis and for implementation. One of the two boards has been connected to the host PC through the PCIe interface, the other one has been connected to the first with four coaxial cables,

two for the Transmitting (TX) channel and two for the Receiving (RX) channel.

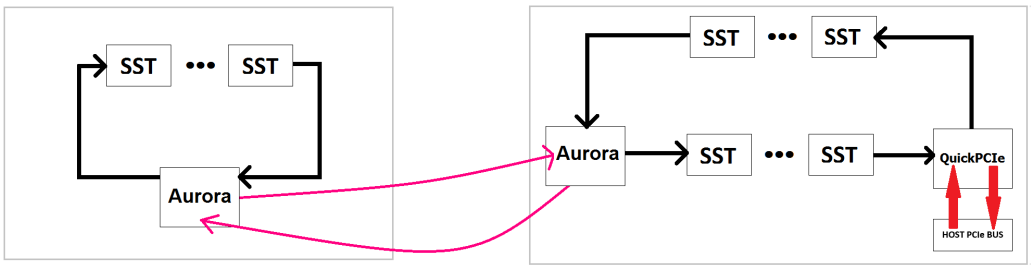


Figure 4.1: A block diagram of the experimental setup.

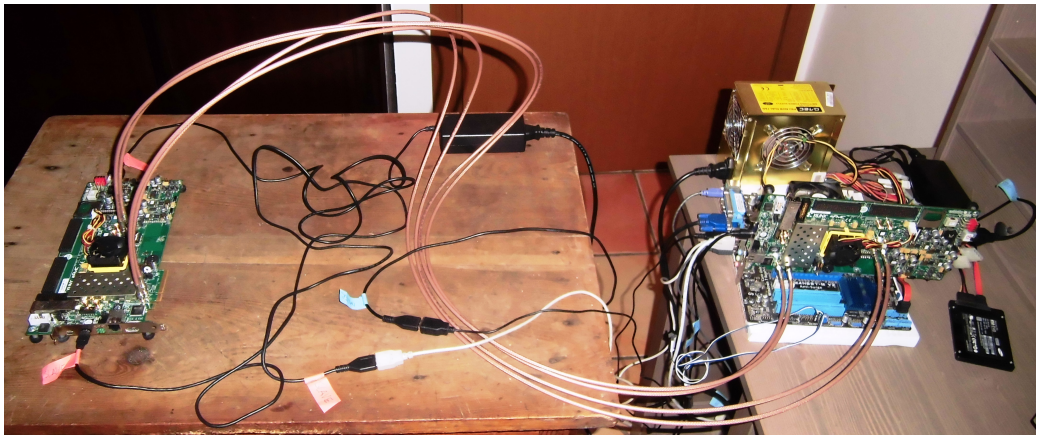
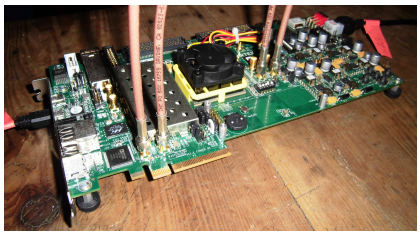


Figure 4.2: The experimental setup for the test session.



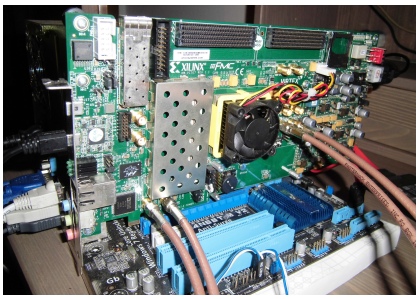


Figure 4.3: The close-ups of the two VC707 boards.

All the tests have been carried on by transmitting a fixed amount of data to the first FPGA and by receiving the computed data, both via the PCIe interface of the host PC. The tests have been executed with an increasing number of SSTs.

Both the transfer of the data and the measurement of the performances have been managed by a custom application based on the Application Programming

Interfaces (APIs) provided by *QuickPCIe* IP [25] core. First, the application initializes the driver for the PCIe interface and scans the host system to discover all the FPGA boards connected on the PCIe bus. Then it locks the available VC707 board and initializes the data to be transferred. When the data are ready, the application starts a DMA transfer from the host memory to the VC707 board. It waits until the entire receive buffer is filled with the data computed by the architecture and then it releases all the locked resources. The overall execution time is measured by exploiting the *gettimeofday()* function. The application computes the throughput of the system and generates a detailed report.

4.2 Test Cases

The architecture has been validated against five benchmarks:

- *jacobi-2D*;
- *seidel-2D*;
- *game-of-life-2D*;
- *jacobi-3D*;
- *heat-3D*.

These benchmarks are based on Iterative Stencil Loops (ISLs), therefore they are suitable for polyhedral-based optimisation. Moreover their hardware implementation allows to easily perform queuing. They represent a small subset of the algorithms used for High Performance Computing (HPC) and they are widely employed in literature.

All the benchmarks have been tested with single precision *floating point* data types, except for *game-of-life-2D*, which has been run with *integer* data. The 2D benchmarks have been fed with a 1000x1000 matrix, while the 3D ones have been fed with a 100x100x100 matrix.

4.2.1 jacobi-2D

The Jacobi method is a popular algorithm for solving Laplace's differential equation on a square domain, regularly discretized. Each array update involves 5 floating point operations. The input matrix used for the test has 1000x1000 elements, therefore for every iteration of the algorithm 4980020 floating-point operations need to be executed.

4.2.2 seidel-2D

The Gauss-Seidel method is an iterative method used to solve a linear system of equations. Each array update involves 9 floating point operations. The input matrix used for the test has 1000x1000 elements, therefore for every iteration of the algorithm 8964036 floating-point operations need to be executed.

4.2.3 game-of-life-2D

John Conway's Game of Life is a well known cellular automaton that models the evolution of two-dimensional universe constituted by a grid of interacting cells. This algorithm has been tested with *integer* data, therefore no floating-point operations have been performed.

4.2.4 jacobi-3D

This is the three-dimensional version of the Jacobi method, already presented in its two-dimensional case. Each array update involves 7 floating point operations. The input matrix used for the test has 100x100x100 elements, therefore for every iteration of the algorithm 6588344 floating-point operations need to be executed.

4.2.5 heat-3D

This method is used to solve the differential equations that model heat conduction into a three-dimensional body. Each array update involves 9 floating point operations. The input matrix used for the test has 100x100x100 elements,

therefore for every iteration of the algorithm 8470728 need to floating-point operations be executed.

4.3 Experimental Results

In this section the experimental results of the previously described benchmarks will be presented. The results must be preceded by an important premise.

The tested architecture is similar to a complex pipeline traversed by a stream of data. The width of its datapath changes across the pipeline from 32bit to 128bit and also the frequency at which the components of the pipeline operate changes from 49Mhz to 125Mhz. Moreover the serial board-to-board interface can be operated only with a single-lane connection and only two boards can be linked together. Hence only a fraction of the maximum available bandwidth can be exploited. Most of the system has a bandwidth of 400MB/s, but there is a bottleneck due to Aurora IP core, which achieves only a bandwidth of 200MB/s on its AXI4 streaming interfaces. The main cause of this behaviour is the impossibility to customize QuickPCIe IP core and Aurora IP core: the first one has been used only as a low-performance evaluation version, the second one needs an external clock generator to achieve higher operating frequencies.

The obtained results however are still valid because the main purposes when performing the tests were:

- to show the scalability of the architecture with the length of the SSTs queue, ensured by the pseudo-linear increase in throughput, in power efficiency and in resource utilization, while remaining with constant bandwidth;
- to show how this approach can outperform the classical HPC systems, concerning power efficiency.

All the results have been compared with their Central Processing Unit (CPU) version, carried on three Intel processors:

- Core i7 2675QM[9]: four cores, eight threads, 22nm lithography, 45W TDP, 2.4GHz;

- Xeon E5-1410[11]: four cores, eight threads, 32nm lithography, 80W TDP, 2.8GHz.
- Xeon E5520[12]: four cores, eight threads, 45nm lithography, 80W TDP, 2.26GHz.

The first processor is a high-end CPU designed with a recent lithography, the second and the third are server class CPUs designed for heavy computations.

The CPU version of each test have been executed with *PLUTO* [23], by applying the *Diamond Tiling* [50] optimization. *OpenMP*[20] has been used in order to exploit all the eight available threads. Moreover the results obtained by *Natale and Sicignano* [7] are included. Their tests have been executed on a single-board architecture running at a 200Mhz frequency and using a 1080x1920 Full-HD matrix. Due to the absence of fast a host-board connection *i.e.* PCIe, all the input data have been pre-loaded into the on-board DDR3 RAM memory and no power has been required by the FPGA transceivers. The time needed to load the data on the board has been neglected while computing the execution time. These results have been chosen not to perform a direct comparison, but to show the performance of the proposed solution when implemented on a working multi-FPGA architecture.

By considering the previously described bandwidth issue, the comparisons can be performed using these metrics:

- Throughput computed as the ratio of the executed Floating Point Operations (FLOPs) to the execution time;
- Power efficiency computed as the ratio of the throughput to the power consumption. Power consumption is estimated by *Vivado Design Suite* [30] when considering the FPGA, while it is the TDP when considering the CPU;
- Resource utilization, for Field Programmable Gate Array (FPGA) only, it is the percentage of the resources used by the considered architecture.
- Frame Rate, computed as the ratio of the length of the SSTs queue to the execution time when considering the FPGA, while it is computed as the ratio

of the number of executed iterations to the execution time when considering the CPU;

All the analysed works do not provide comparable results because they usually do not offer a clear description of the experimental setup and of the executed benchmark. Moreover, the explanation of the results often lacks of some important data, such as the workload and the power consumption. Finally the proposed designs are implemented on custom multi-FPGA systems that are not directly comparable with a real multi-FPGA architecture.

4.3.1 jacobi-2D

The following figures show the performance of the architecture with jacobi-2D SSTs. *Jacobi-2D* is the simplest of the five benchmarks and it is well suited to perform queuing. Therefore, in spite of the limited computational resources, a queue length of 72 has been achieved. In this case the pseudo-linear increase in throughput is clear. Also the power efficiency has a pseudo-linear behaviour and it reaches a value greater than 4.5 GFLOPS/W.

This result is comparable with the efficiency of the fifth *green500 system* [27], although the tests have been executed with single-precision data, in spite of the double-precision data used by the *green500 systems*. The overall performance can be linearly increased beyond the achieved value by adding more SSTs and more FPGA boards to the architecture.

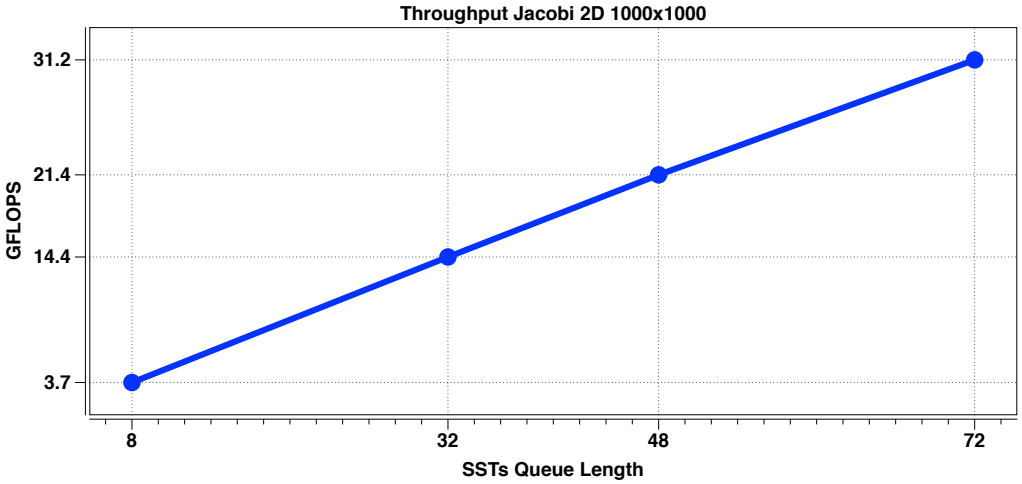


Figure 4.4: jacobi-2D: Throughput (expressed in GFLOPS)

Figure 4.5: jacobi-2D: Power Efficiency (expressed in GFLOPS/W)

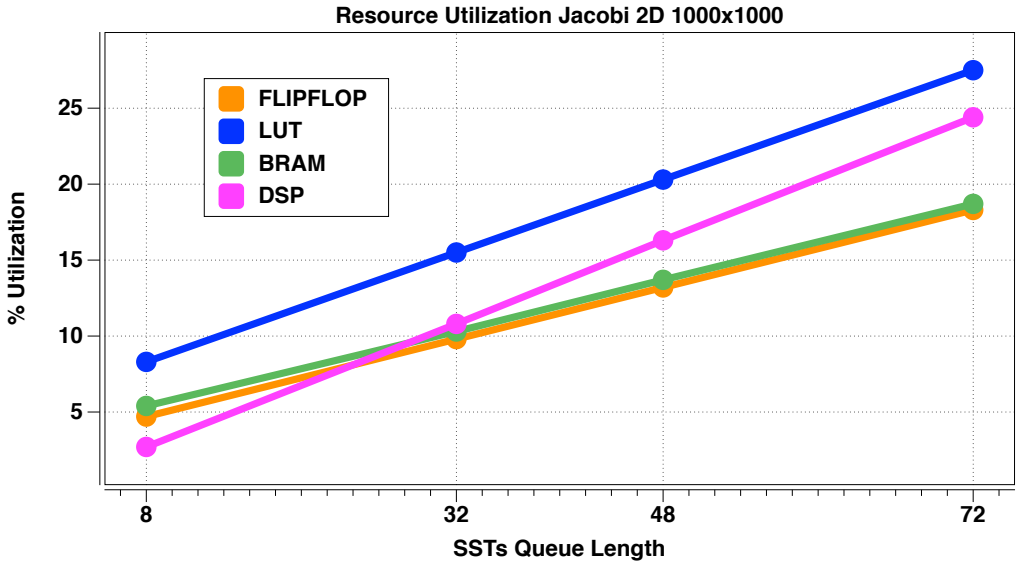


Figure 4.6: jacobi-2D: Resource Utilization (expressed as the percentage of the available resources)

Table 4.1 summarizes the results of all the tests performed with jacobi-2D, and reports also the results achieved by the CPUs and by the design tested by *Natale and Sicignano* [7].

Table 4.1: jacobi-2D

	Throughput	**Power Efficiency**	**FF**	**LUT**	**BRAM**	**DSP**
	(GFLOPS)	(GFLOPS/W)	(%)	(%)	(%)	(%)
Intel Core i7 2675QM	20.458	0.455	-	-	-	-
Intel Xeon E5-1410	19.450	0.243	-	-	-	-
Intel Xeon E5520	12.356	0.154	-	-	-	-
[7] 48 SSTs	23.596	5.775	48.71	39.55	21.50	32.57
8 SSTs	3.674	0.636	4.70	8.31	5.44	2.71
32 SSTs	14.345	2.299	9.81	15.48	10.34	10.86
48 SSTs	21.343	3.319	13.22	20.29	13.69	16.29
72 SSTs	31.204	4.649	18.33	27.48	18.72	24.43

Notice that the proposed architecture outperforms all the three CPUs with respect to the throughput. The design proposed by *Natale and Sicignano*[7] is more power-efficient, because it does not include the PCIe interface and the serial board-board connection.

When considering the power consumption, the CPU version is extremely less efficient than both the FPGA versions. Finally, the proposed design needs less

resources than the one proposed by *Natale and Sicignano*[7].

4.3.2 seidel-2D

The following figures show the performance of the architecture with seidel-2D SSTs. *Seidel-2D* contains spatial dependencies between grid points updates, hence it has no parallelization opportunities. In this case a queue length of 24 SSTs has been achieved, however the system has poor performances.

The throughput shown in figure 4.7 is lower than 1 GFLOPS and the power efficiency, as depicted in figure 4.8, is lower than 0.12 GFLOPS/W. The resource utilization is higher with respect to jacobi-2d.

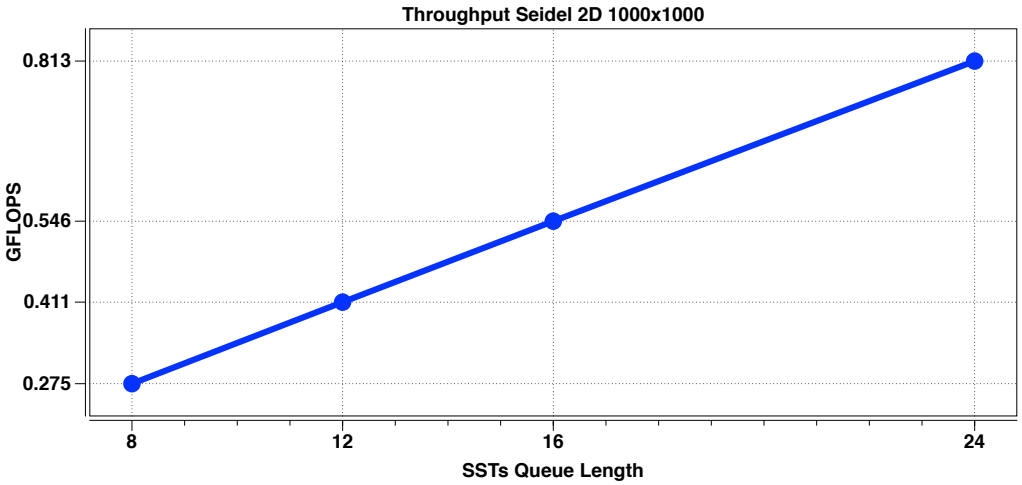


Figure 4.7: seidel-2D: Throughput (expressed in GFLOPS)

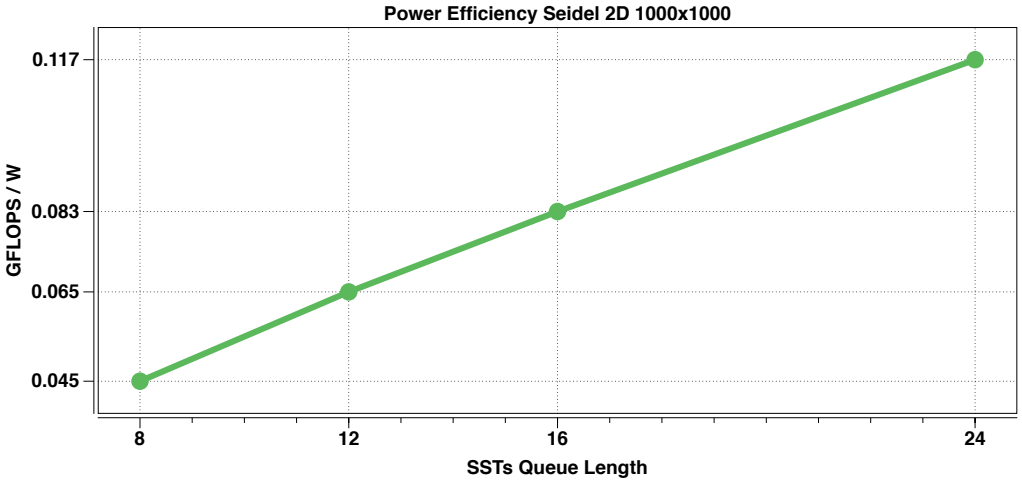


Figure 4.8: seidel-2D: Power Efficiency (expressed in GFLOPS/W)

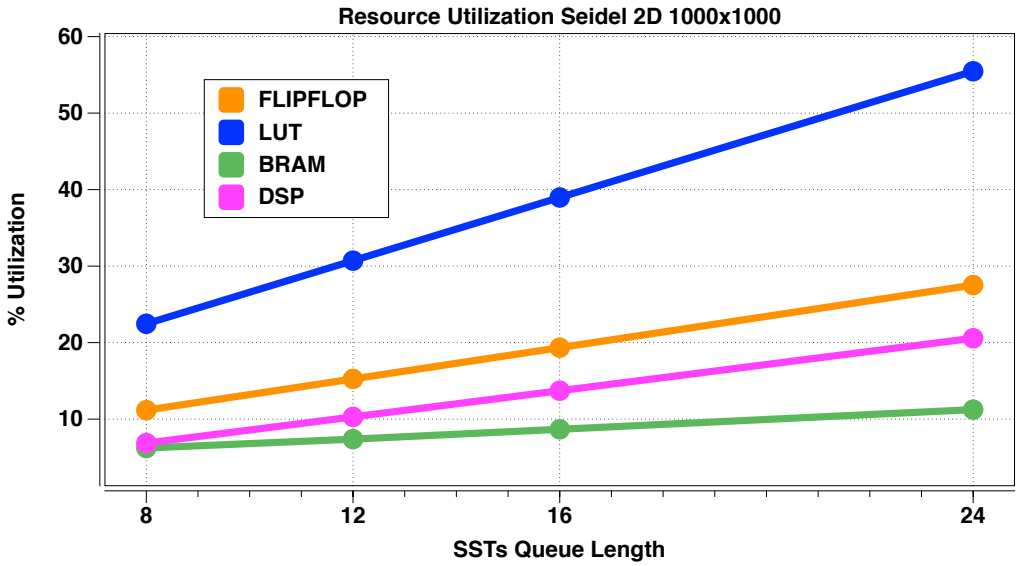


Figure 4.9: seidel-2D: Resource Utilization (expressed as the percentage of the available resources)

Table 4.2 summarizes the results of all the tests performed with seidel-2D, and reports also the results achieved by the CPUs and by the design tested by *Natale and Sicignano* [7].

Table 4.2: seidel-2D

	Throughput	Power Efficiency	FF	LUT	BRAM	DSP
	(GFLOPS)	(GFLOPS/W)	(%)	(%)	(%)	(%)
Intel Core i7 2675QM	0.910	0.020	-	-	-	-
Intel Xeon E5-1410	0.889	0.011	-	-	-	-
Intel Xeon E5520	0.701	0.009	-	-	-	-
[7] 10 SSTs	0.525	0.113	61.60	54.48	8.69	17.14
8 SSTs	0.275	0.045	11.17	22.46	6.22	6.86
12 SSTs	0.411	0.065	15.25	30.71	7.38	10.29
16 SSTs	0.546	0.083	19.34	38.97	8.67	13.71
24 SSTs	0.813	0.117	27.52	55.47	11.24	20.57

Notice that both FPGA architectures outperform the CPU versions of the benckmark with respect to the power efficiency. The proposed design employs less resources than the one proposed by *Natale and Sicignano*[7] and, with a fixed number of boards, allows to achieve a longer SSTs queue.

4.3.3 game-of-life-2D

The following figures show the performance of the architecture with game-of-life-2D SSTs. *Game-of-life-2D* has been tested with integer input data, hence the throughput and the power efficiency could not be computed. To allow the comparison with the other architectures, frame rate has been computed. However this metric does not consider power consumption.

In this case a queue length of 48 SSTs has been achieved, while exploiting only a fraction of the total available resources. The frame rate grows linearly with the length of the queue.

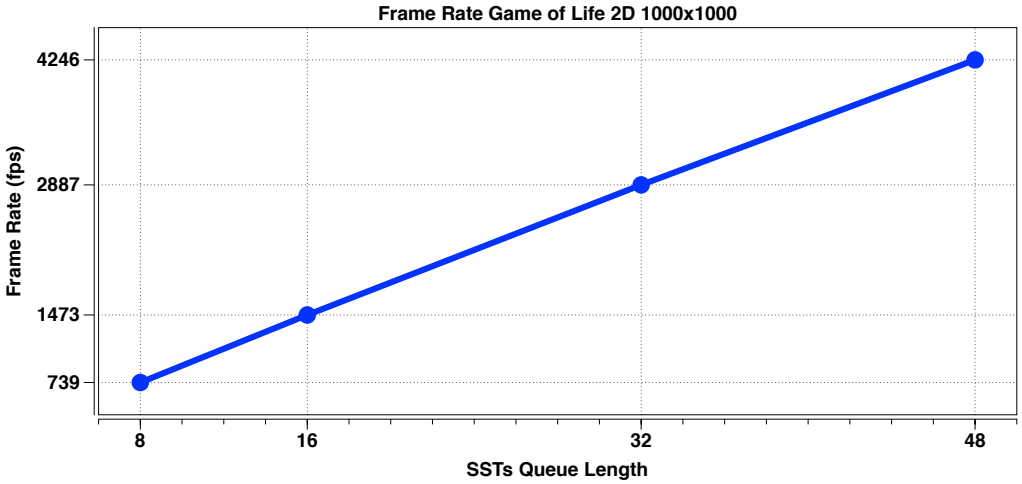


Figure 4.10: game-of-life-2D: Frame Rate (expressed in frame per seconds)

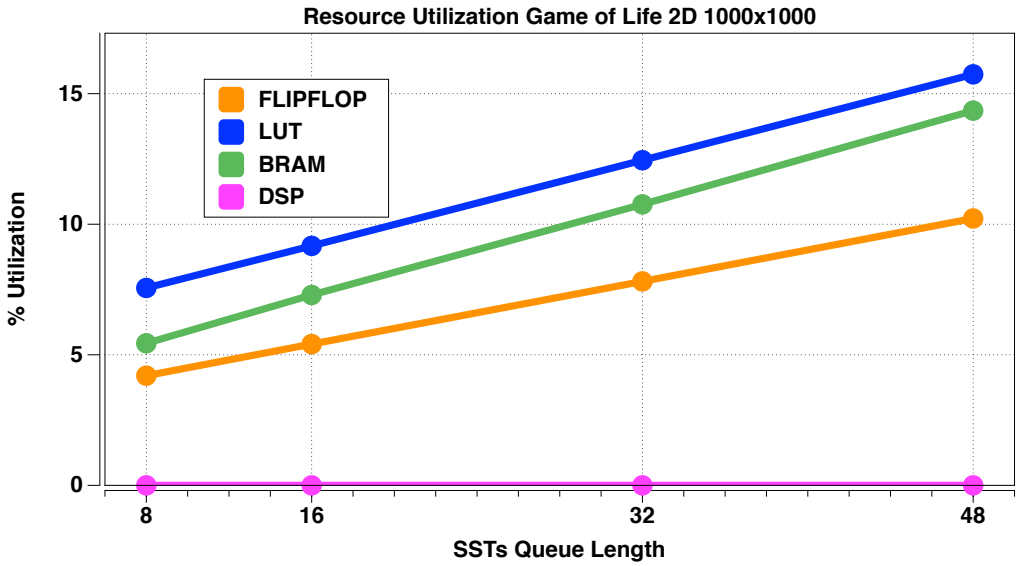


Figure 4.11: game-of-life-2D: Resource Utilization (expressed as the percentage of the available resources)

Table 4.3 summarizes the results of all the tests performed with game-of-life-2D, and reports also the results achieved by the CPUs and by the design tested by *Natale and Sicignano* [7].

Table 4.3: game-of-life-2D

	Frame Rate	FF	LUT	BRAM	DSP
	(fps)	(%)	(%)	(%)	(%)
Intel Core i7 2675QM	372.10	-	-	-	-
Intel Xeon E5-1410	391.35	-	-	-	-
Intel Xeon E5520	216.28	-	-	-	-
[7] 32 SSTs	1502.86	32.51	26.16	15.29	0.00
8 SSTs	738.69	4.20	7.56	5.44	0.00
16 SSTs	1472.89	5.41	9.17	7.29	0.00
32 SSTs	2887.30	7.81	12.45	10.76	0.00
48 SSTs	4246.28	10.22	15.74	14.35	0.00

Notice that the proposed design outperforms all the considered architectures with respect to the frame rate. The resource utilization is, in general, low and no Digital Signal Processing (DSP) unit has been used.

4.3.4 jacobi-3D

The following figures show the performance of the architecture with jacobi-3D SSTs. *Jacobi-3D*, as well as its two-dimensional version, is well suited to perform queuing. Therefore a queue length of 48 has been achieved. As in the previous cases, throughput and power efficiency increase linearly with the length of the queue. Throughput reaches a value greater than 5.5 GFLOPS, while power efficiency never exceeds 0.9 GFLOPS/W. The resources utilization is higher with respect to *jacobi-2d*.

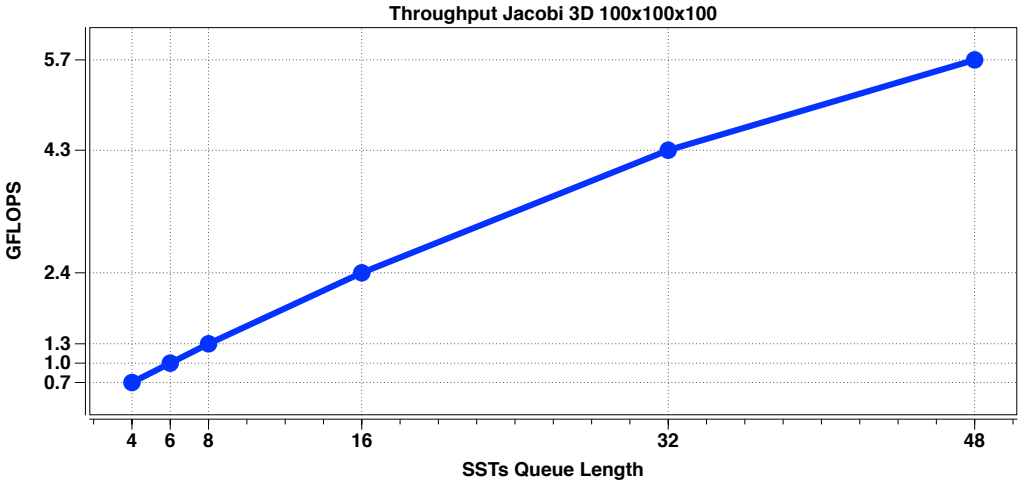


Figure 4.12: jacobi-3D: Throughput (expressed in GFLOPS)

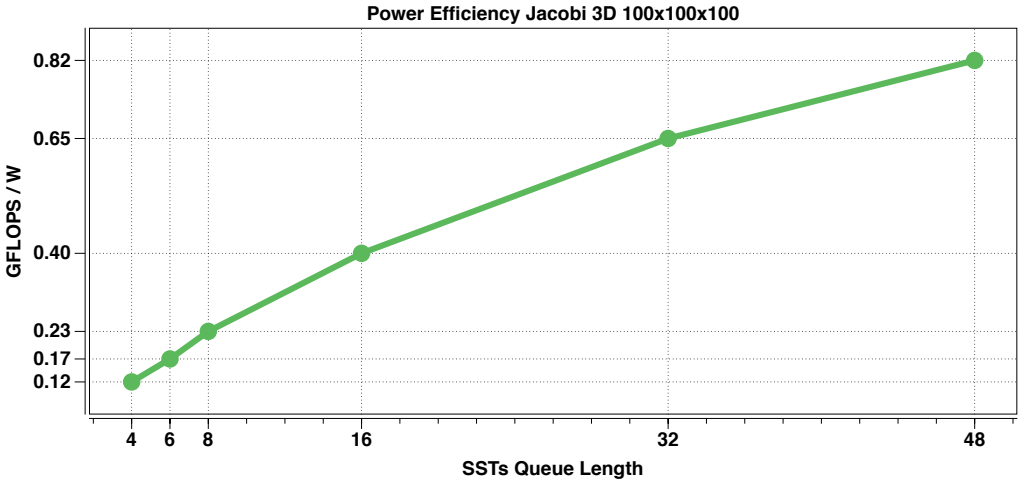


Figure 4.13: jacobi-3D: Power Efficiency (expressed in GFLOPS/W)

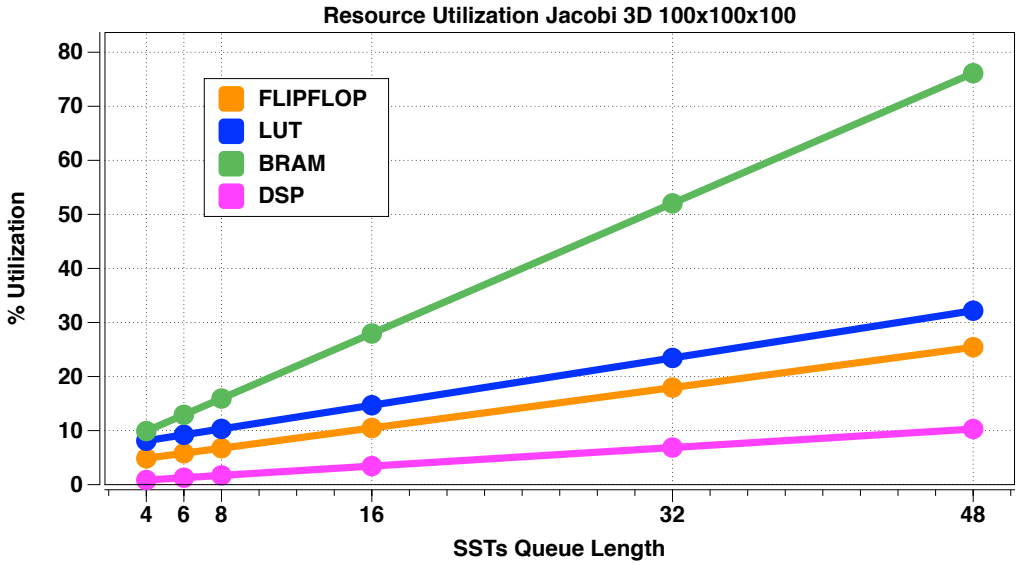


Figure 4.14: jacobi-3D:Resource Utilization (expressed as the percentage of the available resources)

Table 4.4 summarizes the results of all the tests performed with jacobi-3D, and reports also the results achieved by the CPUs and by the design tested by *Natale and Sicignano* [7].

Table 4.4: jacobi-3D

	Throughput	**Power Efficiency**	**FF**	**LUT**	**BRAM**	**DSP**
	(GFLOPS)	(GFLOPS/W)	(%)	(%)	(%)	(%)
Intel Core i7 2675QM	18.982	0.422	-	-	-	-
Intel Xeon E5-1410	16.666	0.208	-	-	-	-
Intel Xeon E5520	12.150	0.152	-	-	-	-
[7] 8 SSTs	2.625	0.802	20.63	16.79	26.94	3.43
4 SSTs	0.679	0.119	4.87	8.12	9.91	0.85
6 SSTs	0.998	0.173	5.81	9.22	12.92	1.28
8 SSTs	1.318	0.226	6.74	10.31	15.93	1.71
16 SSTs	2.429	0.400	10.48	14.69	27.96	3.43
32 SSTs	4.260	0.654	17.95	23.44	52.04	6.86
48 SSTs	5.695	0.823	25.42	32.18	76.12	10.29

Notice that the throughput achieved by the proposed architecture is less than one-third of the throughput obtained with the *Core i7* CPU. The FPGA implementations double the best of the three CPU versions with respects to the power efficiency. Finally, the proposed design needs less resources than the corresponding one proposed by *Natale and Sicignano*[7].

4.3.5 heat-3D

The following figures show the performance of the architecture with heat-3D SSTs. In this case a queue length of 32 has been achieved. As for the previous three-dimensional case, throughput and power efficiency increase linearly with the length of the queue. The obtained results are similar to *jacobi-3D*, although the resources utilization is slightly lower.

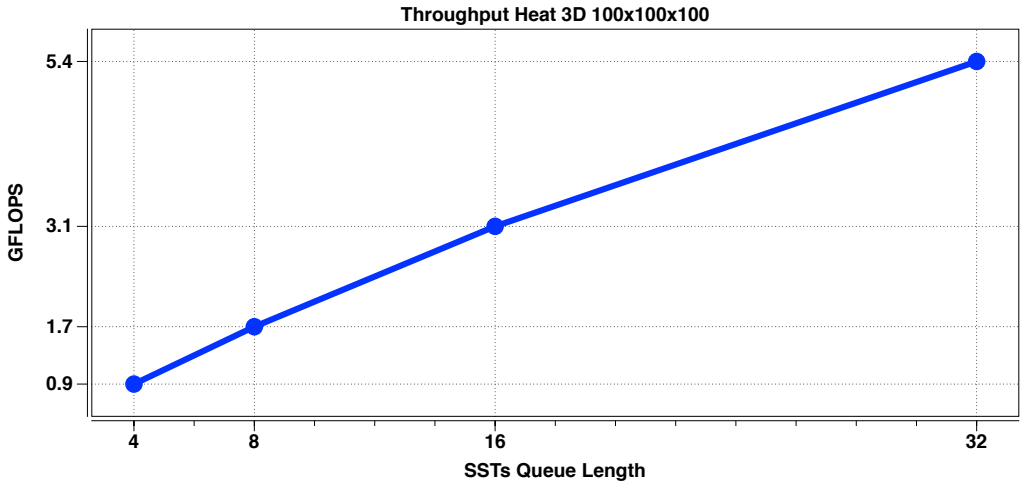


Figure 4.15: heat-3D: Throughput (expressed in GFLOPS)

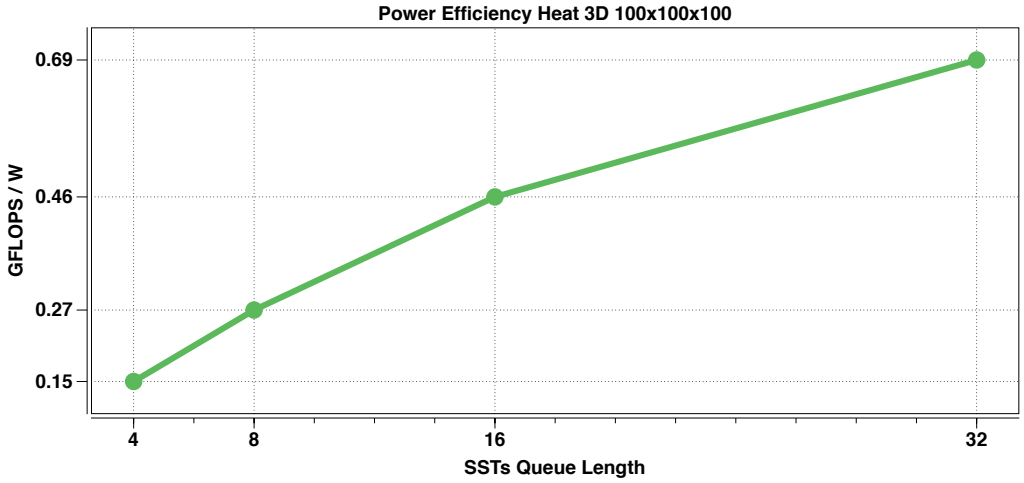


Figure 4.16: heat-3D: Power Efficiency (expressed in GFLOPS/W)

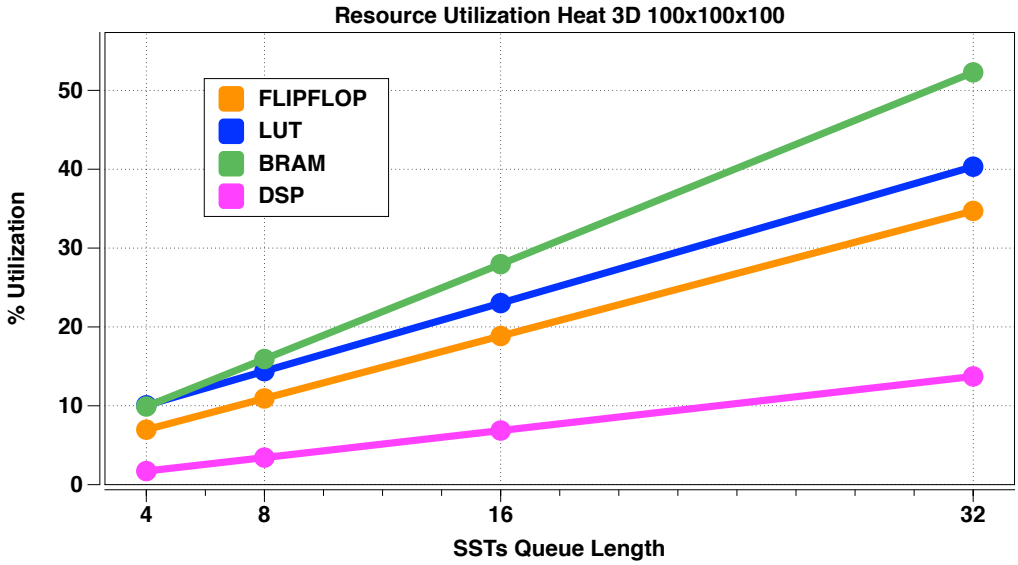


Figure 4.17: heat-3D: Resource Utilization (expressed as the percentage of the available resources)

Table 4.5 summarizes the results of all the tests performed with heat-3D, and reports also the results achieved by the CPUs and by the design tested by *Natale and Sicignano* [7].

Table 4.5: heat-3D

	Throughput	**Power Efficiency**	**FF**	**LUT**	**BRAM**	**DSP**
	(GFLOPS)	(GFLOPS/W)	(%)	(%)	(%)	(%)
Intel Core i7 2675QM	22.988	0.511	-	-	-	-
Intel Xeon E5-1410	16.753	0.209	-	-	-	-
Intel Xeon E5520	12.216	0.153	-	-	-	-
[7] 8 SSTs	3.341	0.946	30.63	24.45	26.94	6.86
4 SSTs	0.869	0.146	6.97	10.06	9.91	1.72
8 SSTs	1.665	0.268	10.93	14.39	15.93	3.43
16 SSTs	3.094	0.458	18.85	23.02	27.96	6.86
32 SSTs	5.421	0.691	34.71	40.32	52.28	13.72

Notice that the throughput achieved by the proposed architecture is less than one-fourth of the throughput obtained with the *Core i7* CPU. With respect to the power efficiency, the proposed design reaches a value of 0.69 GFLOPS/W, which is slightly higher than the value obtained by the best of the three CPUs.

4.4 A Quantitative Model for the Power Efficiency

This section presents a quantitative model that shows how the power efficiency of the proposed architecture increases with the number of used FPGA boards and with the length of the SSTs queue. Indeed, it provides an approximated value of the power efficiency for each configuration of the proposed architecture. The model has been built in order to validate the obtained experimental results and to predict their trend for systems larger than the implemented two-board architecture.

4.4.1 Building the Model

Before illustrating the model, it is necessary to explain which variables and which parameters have been used to build the model and how they have been computed.

The model is expressed by an equation, that is a function of two variables:

- "n" : number of SSTs included into the considered architecture configuration;
- "N" : number of intermediate boards of the considered architecture configuration. It does not include the first and the last boards of the queue;

The model includes also five parameters, that depend on the considered architecture:

- "ω" [FLOP] : number of FLOPs computed by each SST instantiated into the considered architecture configuration;
- "τ" [s] : execution time of the considered architecture configuration. It is computed as the average execution time of the tested configurations;
- "π" [W] : power consumption of each host-board PCIe interface included into the considered architecture configuration;
- "α" [W] : power consumption of each board-board Aurora interface included into the considered architecture configuration;

- "σ" [W] : power consumption of each SST included into the considered architecture configuration.

In order to obtain the equation of the model it is necessary to recall the general equation of the power efficiency:

$$PowerEfficiency = \frac{Throughput}{PowerConsumption} = \frac{\frac{PerformedComputations}{ExecutionTime}}{PowerConsumption} \quad (4.1)$$

When considering the proposed architecture, we have:

- PerformedComputations = n * ω, because the data are processed by the SSTs only;
- ExecutionTime = τ;
- PowerConsumption = 2*(π+α)+N*α+n*σ. The first contribution is due to the first and to the last board of the queue, the second one accounts for the intermediate boards, the last contribution is due to SSTs instantiated into the architecture.

Finally, by substituting into the equation of the power efficiency, the general equation for the model can be obtained:

$$PE(n, N) = \frac{\frac{n*\omega}{\tau}}{2*(\pi+\alpha)+N*\alpha+n*\sigma} \quad (4.2)$$

4.4.2 Power Efficiency Trends

The obtained equation is a function of two variables, therefore two cases have been analysed in order to show the trend followed by the power efficiency:

- in the first case a system with no intermediate boards has been considered. The length of the SSTs queue can be increased;
- in the second case each board of the system can host only a SST. The number of intermediate boards can be increased.

In both cases the values assigned to the parameters and the values obtained from the experimental results are of the same order of magnitude.

In the first case it is N=0, therefore the equation of the model is:

$$PE(n) = \frac{\frac{n*\omega}{\tau}}{2*(\pi+\alpha)+n*\sigma} \tag{4.3}$$

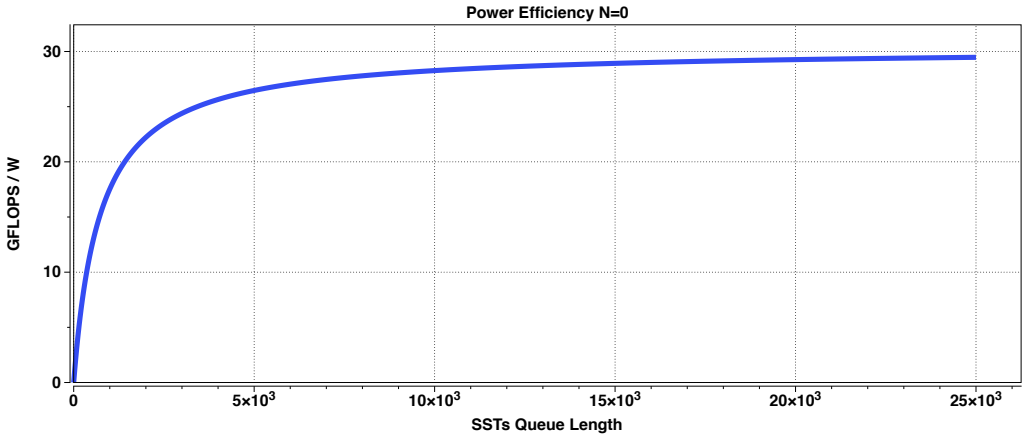


Figure 4.18: Power Efficiency trend of a two-boards system when increasing the SSTs queue length

As shown in figure 4.18, the power efficiency grows quickly with the length of the SSTs queue, up to a theoretical limit, that can be computed by considering n->∞:

$$PE_{limit} = \frac{\frac{\omega}{\tau}}{\sigma} = \frac{\omega}{\tau*\sigma} \tag{4.4}$$

In the second case it is n=2+N, because the first board and the last board host one SST each and one SST is instantiated on each intermediate board.

Therefore the equation of the model is:

$$PE(N) = \frac{\frac{(2+N)*\omega}{\tau}}{2(\pi+\alpha)+N\alpha+(2+N)\sigma} = \frac{\frac{N*\omega}{\tau}+\frac{2*\omega}{\tau}}{N(\alpha+\sigma)+2(\pi+\alpha+\sigma)} \tag{4.5}$$

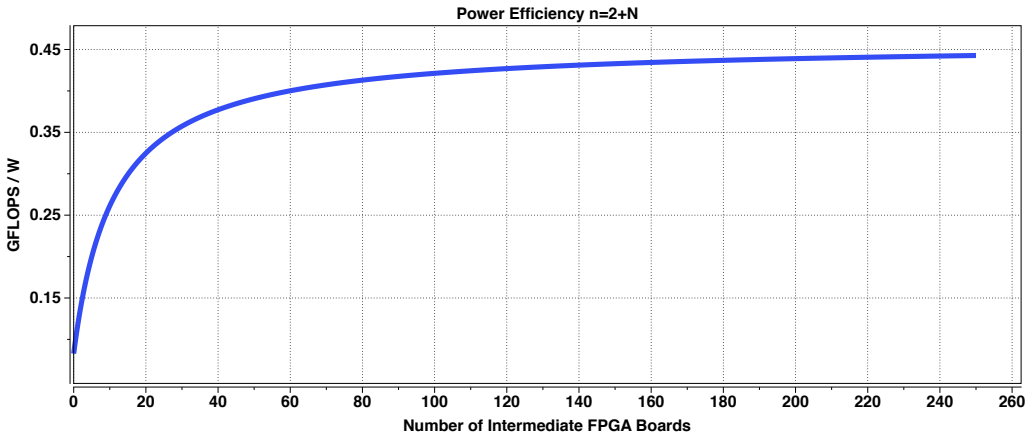


Figure 4.19: Power Efficiency trend of a system with a single SST on each board when increasing the number of intermediate FPGA boards

As shown in figure 4.19, the power efficiency grows quickly with the queue of FPGA boards, up to a theoretical limit, that can be computed by considering N->∞:

$$PE_{limit} = \frac{\frac{\omega}{\tau}}{\alpha+\sigma} = \frac{\omega}{\tau*(\alpha+\sigma)} \tag{4.6}$$

Figure 4.20 shows that the implemented architecture can be scaled up to a theoretical limit both by increasing the length of the SSTs queue and by growing the queue of FPGA boards. Notice that no constraint has been posed to the

maximum number of boards that can be plugged into a cluster node and to the maximum number of SSTs that can fit into an FPGA.

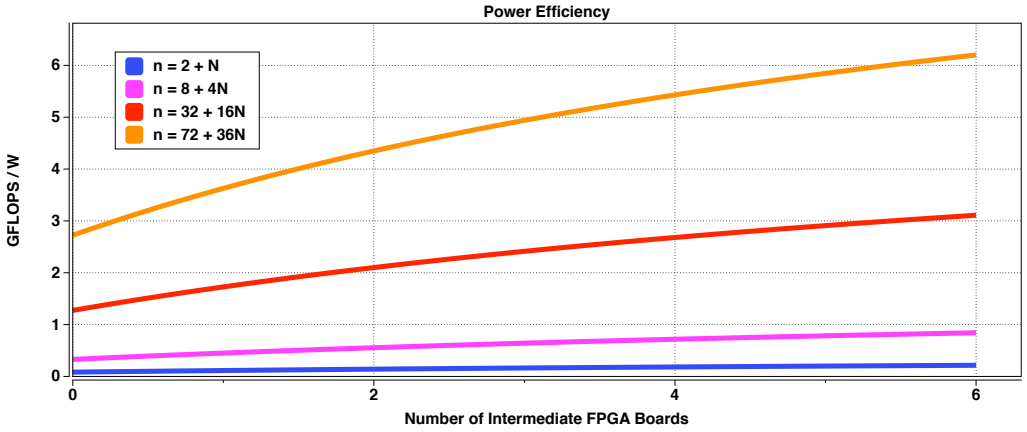


Figure 4.20: Power Efficiency trend when increasing the number of intermediate FPGA boards for different SSTs queue lengths

4.4.3 Model Validation

In this section the quantitative model is compared with the experimental results obtained with two benchmarks: Jacobi2D and Heat3D. In order to perform a proper comparison, the equation of the model must be slightly modified, because only one of the two FPGA boards features a PCIe interface.

Starting from the general equation of the model,

$$PE(n, N) = \frac{\frac{n*\omega}{\tau}}{2*(\pi+\alpha)+N*\alpha+n*\sigma} \tag{4.7}$$

and considering a two-boards system, with only one board connected on the PCIe bus, we have:

- no intermediate boards, hence N=0;
- one PCIe interface only, hence PowerConsumption = π+2*α+n*σ.

Therefore the following equation can be obtained:

$$PE(n) = \frac{\frac{n*\omega}{\tau}}{\pi + 2 * \alpha + n * \sigma} \tag{4.8}$$

The tests have been performed with two VC707 boards, by employing Quick-PCIe and Aurora IP cores. Therefore the power consumed by the PCIe interface and by the board-board serial link is known, hence:

- π = 4.5 W;
- α = 0.977 W.

Jacobi 2D SSTs

When considering Jacobi 2D SST, the following values are assigned to the parameters:

- $\omega = 5 * 10^6$ FLOPs;
- τ = 0.011 s;
- σ = 0.015 W.

Therefore the equation in this case is:

$$PE(n)_{Jac2D} = \frac{n * 4.55 * 10^8}{6.45 + n * 0.015} \tag{4.9}$$

The comparison with the experimental results is depicted in figure 4.21.

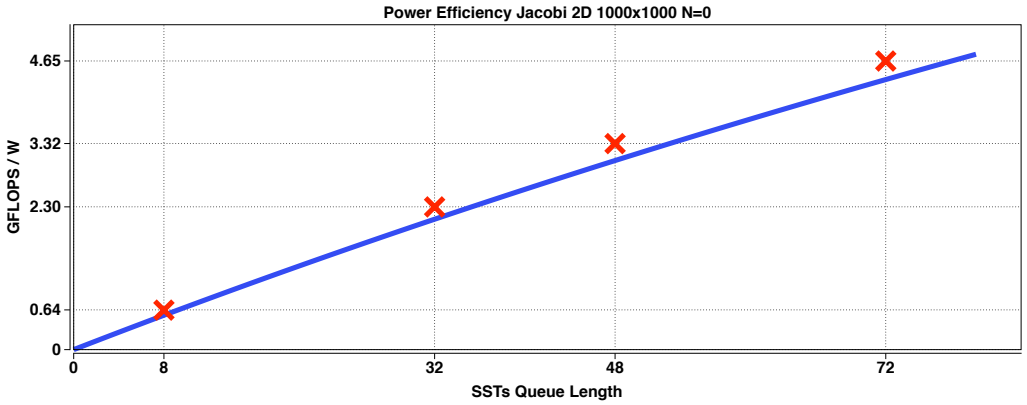


Figure 4.21: Comparison between the experimental results (red X marks) and the predicted values (blue line) when considering Jacobi2D SSTs

Heat 3D SSTs

When considering Heat 3D SST, the following values are assigned to the parameters:

- $\omega = 8.5 * 10^6$ FLOPs;
- $\tau = 0.043$ s;
- $\sigma = 0.067$ W.

Therefore the equation in this case is:

$$PE(n)_{Heat3D} = \frac{n * 1.98 * 10^8}{6.45 + n * 0.067} \tag{4.10}$$

The comparison with the experimental results is depicted in figure 4.22.

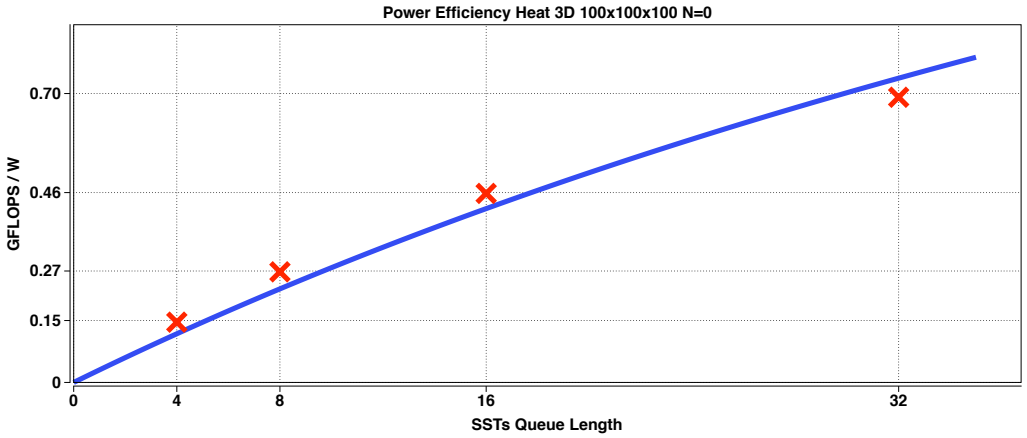


Figure 4.22: Comparison between the experimental results (red X marks) and the predicted values (blue line) when considering Heat3D SSTs

Notice that the proposed model fits well both the analysed benchmarks. However, the power efficiency trend is predicted more precisely for the Jacobi2D benchmark than for the Heat3D benchmark. This behaviour can be explained by considering the values assigned to the parameters:

- in the case of Jacobi2D the execution time does not increase clearly when the SSTs queue grows from 8 to 72, therefore the average execution time assigned to "τ" coincides with the execution time for each experimental value;
- when Heat3D is considered, the execution time changes significantly when the SSTs queue grows from 4 to 32, therefore the average execution time assigned to "τ" only estimates the execution time for each experimental value.

Moreover, some minor contributions to the power consumptions have be neglected, due to inability to evaluate them accurately.

Finally, the provided comparisons prove the high scalability of the architecture. They also show how the practical limitation posed by the technology to the maximum length of the SSTs queue does not allow to reach the theoretical limitation predicted by the model. Therefore the power efficiency of the proposed architecture always grows along the pseudo-linear portion of the trend predicted by the model.

5

Conclusions and Future Work

5.1 Conclusions

In this work a cluster node architecture for the *exaFPGA project* [6] has been presented. The proposed design enables the use of the PCIe interface and of the Aurora serial link on the VC707 board. It allows to easily interface a FPGA board with a host PC and with more FPGA boards in order to build a working multi-FPGA HPC system.

The implemented design consists of a complex full-streaming pipeline spanning over all the boards included into the system. No on-board DDR3 RAM and no CPU IP core is present in this design and no firmware is run on the boards. Data can be transferred at a high speed thanks to the use of the FPGA transceivers both for the PCIe connection and for the serial link. The computation is managed by a custom application running on the host PC, no further software needs to be developed. The SSTs proposed by *Natale and Sicignano* [7] can easily be queued and included into the pipeline.

Due to its features, the proposed architecture can be scaled by adding more FPGA boards and by increasing the SST queue length, as explained in chapter 3 and in chapter 4. A quantitative model has been built in order to predict the power efficiency trend followed by a multi-FPGA system. This model has been validated against the experimental results, showing a good compliance to the observed power efficiency trends.

Experimental results show that the resources utilization grows linearly with the length of the SST queue, only a fraction of the total available resources is reserved for non-computing logic *i.e.* host-board and board-board communication. Throughput can be linearly increased without any strong limitation except for the maximum achievable SSTs queue length and for the maximum number of FPGA boards that can fit into a cluster node. The fast serial board-board communication allows to add more boards and more SSTs without adding any bottleneck to the system. Power efficiency grows linearly with the SSTs queue and it is also expected to increase with the number of boards included into the system, due to the high power consumption of the transceivers used only for the PCIe interface of the first and of the last board of the architecture. The performed comparisons show how the proposed architecture allows to reach the performance of the most power efficient HPC systems[27].

5.2 Future Work

There are a number of possible developments of the proposed work.

The first is to improve the overall performance in order to push the system to its limit. This can be achieved by increasing the available resources for the design phase and for the test phase *i.e.* by employing a professional workstation, by purchasing commercial Intellectual Property (IP) cores in spite of their evaluation versions and by exploiting an external generator for the board-board serial connection. These improvements would allow to balance all the pipeline steps and to increase the overall throughput. Furthermore, all the resources on the FPGA could be used and a system with more than two boards could be built, thus increasing the achievable throughput and the power efficiency.

Another future work consists in switching from a single precision to a double precision data type. This would allow a better exploitation and a better balance of the datapath, thus increasing the overall performances. Moreover it would be easier to compare the experimental results of the proposed architecture with the ones presented in the literature.

Currently, only single-input SSTs are supported, so another further development is the support for multi-input SSTs. This could be achieved by implementing a custom protocol to transmit the multiple data for the SST over the single path traversing the pipeline. Obviously the structure of the SST must be modified in order to expose a single input stream and a single output stream. A feasible solution could be to add a demultiplexing and a multiplexing stage to the input and to the output of each SST, respectively. This solution would allow to manage multi-input SSTs without any change to the overall architecture. However this enhancement could cause a serious performance degradation due to the demultiplexing/multiplexing stages and to the reduction of the payload.

The VC707 board permits to use only an Aurora IP core in single lane configuration. Therefore the proposed design could also be implemented by employing an high-end FPGA board which provides a wider serial board-board link. Moreover an increasing number of Aurora IP cores could be instantiated on a FPGA, thus allowing the use of multiple serial channels. This could be an important enhancement if considering the design of a more complex pipeline featuring parallel paths in place of the single one already implemented.

The proposed work has been carried on within the *exaFPGA project* [6], therefore a future development is the integration of the architecture with the already implemented software tool, which automatically generates a SST starting from the C language specification of an algorithm. This could be achieved by wrapping the architecture into multiple IPs cores, one for each board included into the system. The tool would automatically connect the SSTs queue to the generated IP cores, by exploiting the exposed AXI4 streaming interfaces.

Finally, there are two more future enhancements: the first is the employment of partial reconfiguration in order to make the system more flexible and to better the management of the resources. The second is the reinforcement of the system *i.e.* the use of redundant parts, the adoption of Error-Correcting Codes (ECCs), in order to improve the dependability of the proposed solution and to lower the error probability.

Bibliography

[1] 7 Series Gen2 Integrated Block for PCI Express (PCIe) . http://www.xilinx.com/products/intellectual-property/axi_pcie_gen3.html.

[2] AXI Bridge for PCI Express (PCIe) Gen3 Subsystem. http://www.xilinx.com/products/intellectual-property/7_series_pci_express_block.html.

[3] Clan - a polyhedral representation extractor for high level programs. http://icps.u-strasbg.fr/~bastoul/development/clan/docs/clan.html.

[4] Convey Computer. http://www.conveycomputer.com/.

[5] EPEE Library. http://cecaraw.pku.edu.cn/Eng_EPEE.html.

[6] exaFPGA Project. http://exafpga.necst.it/.

[7] Giuseppe Natale, Carlo Sicignano. On how to Accelerate Iterative Stencil Loops: A Scalable Streaming-based Approach. https://www.politesi.polimi.it/handle/10589/106951.

[8] Human Brain Project. https://www.humanbrainproject.eu.

[9] Intel Core i7 2675QM. http://ark.intel.com/it/products/53470/Intel-Core-i7-2675QM-Processor-6M-Cache-up-to-3_10-GHz.

[10] Intel Introduces Configurable Intel Atom-based Processor. http://newsroom.intel.com/docs/DOC-1512.

[11] Intel Xeon E5-1410.
http://ark.intel.com/products/67417/Intel-Xeon-Processor-E5-1410-10M-Cache-2_8-GHz.

[12] Intel Xeon E5520.
http://ark.intel.com/it/products/40200/Intel-Xeon-Processor-E5520-8M-Cache-2_26-GHz-5_86-GTs-Intel-QPI.

[13] Intel Xeon Phi Product Family.
http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html.

[14] Maxeler Technologies.
https://www.maxeler.com.

[15] Microsoft Catapult project.
http://research.microsoft.com/en-us/projects/catapult.

[16] NetFPGA Project.
http://netfpga.org/.

[17] NVIDIA CUDA.
http://www.nvidia.com/object/cuda_home_new.html.

[18] NVIDIA Fermi.
http://www.nvidia.it/object/fermi_architecture_it.html.

[19] NVIDIA Kepler.
http://www.nvidia.com/object/nvidia-kepler.html.

[20] OpenMP.
http://openmp.org/wp/.

[21] Pico Computing.
http://picocomputing.com/.

[22] PLDA.
https://www.plda.com/.

[23] PLUTO.
http://pluto-compiler.sourceforge.net/.

[24] POCC the Polyhedral Compiler Collection.
pocc.sourceforge.net/.

[25] QuickPCIe.
https://www.plda.com/products/fpga-ip/xilinx/fpga-ip-pcie/quickpcie-ep-expert-xilinx.

[26] Riffa 2.0.
https://sites.google.com/a/eng.ucsd.edu/matt-jacobsen/about/publications.

[27] The Green 500 List.
http://www.green500.org/.

[28] The Top 500 List.
http://www.top500.org/.

[29] VC707 board.
http://www.xilinx.com/products/boards-and-kits/ek-v7-vc707-g.html.

[30] Vivado Design Suite.
http://www.xilinx.com/products/design-tools/vivado.html.

[31] Vivado HLS.
http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[32] Xilinx Aurora 64b/66B.
http://www.xilinx.com/products/design_resources/conn_central/grouping/aurora.htm.

[33] Xilinx AXI Chip2Chip.
http://www.xilinx.com/products/intellectual-property/axi-chip2chip.html.

[34] Xilinx AXI DMA Controller.
http://www.xilinx.com/products/intellectual-property/axi_dma.html#overview.

[35] Xilinx Memory Interface Generator (MIG).
http://www.xilinx.com/products/intellectual-property/mig.html.

[36] Xilinx MicroBlaze Soft Processor Core.
http://www.xilinx.com/tools/microblaze.htm.

[37] Xillybus.
http://xillybus.com/.

[38] Zynq-7000 All Programmable SoC.
http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[39] Zynq UltraScale+ MPSoC.
http://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.

[40] Intel Atom Processor E6x5C Series-Based Platform for Embedded Computing. Intel Corporation, 2010.

[41] Rethink Flexibility with a Configurable Intel Atom Processor. Intel Corporation, 2010.

[42] NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210. NVIDIA Corporation, 2014.

[43] Top ten exascale research challenges. U.S. Department of Energy, Office of Science, 2014.

[44] High performance computing in the eu: Progress on the implementation of the european hpc strategy. European Union, 2015.

[45] High performance computing strategic plan 2015-2020. National Oceanic and Atmosferic Administration. U.S. Department of Commerce, 2015.

[46] Zynq-7000 All Programmable SoC Technical Reference Manual. Xilinx Inc., 2015.

[47] M. Benabderrahmane, L. Pouchet, A. Cohen, and C. Bastoul. The polyhedral model is more widely applicable than you think. CC'10/ETAPS'10 Proceedings of the 19th joint European conference on Theory and Practice of Software, international conference on Compiler Construction, 2010.

[48] George Chrysos. Intel Xeon Phi Coprocessor-the Architecture. Intel Corporation, 2012.

[49] S. Embid Droz and R. Rodriguez Torrado. High performance computing oil and gas industry. the way to open new opportunities. Repsol Upstream Technology Unit.

[50] Bandishti et al. Tiling Stencil Computations to Maximize Parallelism. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and*

Analysis, SC '12, pages 40:1–40:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[51] D. E. Shaw et al. Anton, a Special-Purpose Machine for Molecular Dynamics Simulation. D. E. Shaw Research, New York, USA, 2007.

[52] D. E. Shaw et al. Anton 2: Raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. D. E. Shaw Research, New York, USA, 2014.

[53] Lindholm et al. NVIDIA TESLA: A UNIFIED GRAPHICS AND COMPUTING ARCHITECTURE. IEEE Computer Society, 2008.

[54] R. Pordes et al. New science on the open science grid. IOP Publishing Ltd, 2008.

[55] Erol Gelenbe and Yves Caseau. The impact of information technology on energy consumption and carbon emissions. ACM, 2015.

[56] Peter N. Glaskowsky. NVIDIA's Fermi: The First Complete GPU Computing Architecture. 2009.

[57] Oliver Pell Michael J Flynn and Oskar Mencer. Dataflow Supercomputing. IEEE Computer Society, 2012.

[58] David Patterson. The Top 10 Innovations in the New NVIDIA Fermi Architecture, and the Top 3 Next Challenges. 2009.

[59] S. Pop, A. Cohen, C. Bastoul, G. Silber, N. Vasilache, and S. Girbal. Graphite: Polyhedral analyses and optimizations for gcc. GCC Developers' Summit, 2006.

[60] Andrew Putnam, Adrian Caulfield, Eric Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, Jim Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *41st Annual International Symposium on Computer Architecture (ISCA)*, June 2014.

[61] M. C. Schatz. High performance computing for dna sequence alignment and assembly. University of Maryland, 2010.

[62] Jeff Stuecheli. POWER8. IBM Power Systems, 2013.

[63] Michael B. Taylor. Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse. ACM, 2012.

[64] Wim Vanderbauwhede and Khaled Benkrid. High-performance computing using fpgas. Springer Science+Business Media, 2014.

[65] Bruce Wile. Coherent Accelerator Processor Interface (CAPI) for POWER8 Systems. IBM Systems and Technology Group, 2014.

[66] Xilinx. Aurora 64B/66B v10.0 - LogiCORE IP Product Guide, 2015.

[67] S. A. Zenios. High-performance computing in finance: The last 10 years and the next. Elsevier Science B.V, 1999.

[68] Noa Zilberman, Yury Audzevich, G. Adam Covington, and Andrew W. Moore. Netfpga sume: toward 100 gbps as research commodity. IEEE Computer Society, 2014.

December 2, 2015

Document typeset with LaTeX