UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXVI CICLO DEL DOTTORATO DI RICERCA IN INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

HyperFPGA: SoC-FPGA Cluster Architecture for Supercomputing and Scientific applications

Settore scientifico-disciplinare: ING-INF/01 Electronics

DOTTORANDO:	**Werner Oswaldo Florian Samayoa**	[illegible]
COORDINATORE:	**Fulvio Babich**	[illegible]
SUPERVISORE DI TESI UNITS:	**Sergio Carrato**	[illegible]
SUPERVISORE DI TESI ICTP:	**Maria Liz Crespo**	[illegible]

ANNO ACCADEMICO 2022/2023

This page intentionally left blank.

Acknowledgments

At the end of this degree, I can only look back and appreciate all the people, gifts, and experiences that made this an invaluable journey.

First, I would like to dedicate this work to my maker and Savior Jesus Christ for giving me the abilities, training, passions, empowerment, and determination to run this race.

I would like to express my heartfelt gratitude to my family, friends, supervisors, and colleagues who supported and inspired me throughout my endeavor to complete this Ph.D. thesis.

I want to thank my family for their constant love and encouragement.

To my friends who stood by me through the highs and lows, offering wise advice and careless joy when appropriate.

I am also indebted to my supervisors who generously provided guidance, wisdom, knowledge, and patience.

I extend my most sincere thanks to the ICTP Multidisciplinary Laboratory for betting on me from the beginning and supporting me along with the University of Trieste.

Finally, I would like to acknowledge the countless researchers whose work served as the foundation for this thesis. Their contributions to the field have been invaluable in shaping my own understanding and ideas.

Thank you all for being a part of this significant milestone in my life.

This page intentionally left blank.

Abstract

Since their inception, supercomputers have addressed problems that far exceed those of a single computing device. Modern supercomputers are made up of tens of thousands of CPUs and GPUs in racks that are interconnected via elaborate and most of the time ad hoc networks [1]. These large facilities provide scientists with unprecedented and ever-growing computing power capable of tackling more complex and larger problems [2, 3]. In recent years, the most powerful supercomputers have already reached megawatt power consumption levels, an important issue that challenges sustainability and shows the impossibility of maintaining this trend.

With more pressure on energy efficiency [4, 5], an alternative to traditional architectures is needed. Reconfigurable hardware, such as FPGAs, has repeatedly been shown to offer substantial advantages over the traditional supercomputing approach with respect to performance and power consumption [6, 7, 8]. In fact, several works that advanced the field of heterogeneous supercomputing using FPGAs are described in this thesis [9]. Each cluster and its architectural characteristics can be studied from three interconnected domains: network, hardware, and software tools, resulting in intertwined challenges that designers must take into account. The classification and study of the architectures illustrate the trade-offs of the solutions and help identify open problems and research lines, which in turn served as inspiration and background for the HyperFPGA.

In this thesis, the HyperFPGA cluster is presented as a way to build scalable SoC-FPGA platforms to explore new architectures for improved performance and energy efficiency in high-performance computing, focusing on flexibility and openness. The HyperFPGA is an open-source modular platform based on a SoM that includes power monitoring tools with high-speed general-purpose interconnects to offer a great level of flexibility and introspection. By exploiting the reconfigurability and programmability offered by the HyperFPGA infrastructure, which combines FPGAs and CPUs, with high-speed general-purpose connectors, novel computing paradigms can be implemented. A custom Linux OS and drivers, along with a custom script for hardware definition, provide a uniform interface from application to platform for a programmable framework that integrates existing tools. The development environment is demonstrated using the N-Queens problem, which is a classic benchmark for evaluating the performance of parallel computing systems. Overall, the results of the HyperFPGA using the N-Queens problem highlight the platform's ability to handle computationally intensive tasks and demonstrate its suitability for its use in supercomputing experiments.

Contents

List of Figures

List of Tables

Acronyms

μP	Microprocessor.
ADC	Analog to Digital Converter.
AI	Artificial Intelligence.
AO	Adaptive Optics.
API	Application Programming Interface.
APU	Application Processing Unit.
ASIC	Application Specific Integrated Circuit.
ATF	Arm Trusted Firmware.
BEE	Berkeley Emulation Engine.
BMC	Baseboard Management Controller.
BPU	BEE Processing Unit.
BSP	Board Support Package.
CAD	Computer Assisted Design.
CB	Concentrator Board.
CE	Computational Element.
ComBlock	Communication Block.

COTS Commercial Off-The-Shelf.

CPLD Complex Programmable Logic Device.

CPU Central Processing Unit.

CRC Cyclic Redundancy Check.

CSU Configuration Security Unit.

CU Computational Unit.

DAQ Data Acquisition System.

DB Daughter Board.

DC Data Center.

DCDC Direct-Current to Direct-Current.

DDR Dual Data Rate.

DFM Data Flow Module.

DIMM Dual Inline Memory Module.

DMA Direct Memory Access.

DPR Dynamic Partial Reconfiguration.

DSE Design Space Exploration.

DSM Distributed-Shared Memory.

DSP Digital Signal Processor.

DWCE Digital Wireless Channel Emulator.

EDA Electronic Design Automation.

FaaS FPGA as a Service.

FB	Functional Block.
FHPCA	FPGA High-Performance Computing Alliance.
FIFO	First-In First-Out.
FIR	Finite Impulse Response.
FMC	FPGA Mezzanine Connector.
FPGA	Field Programmable Gate Array.
FSBL	First Stage Boot Loader.
FTP	File Transfer Protocol.
GbE	Gigabit Ethernet.
GPGPU	General-Purpose Graphics Processing Unit.
GPIO	General Purpose Input Output.
GPU	Graphics Processing Unit.
GTH	Gigabit Transceiver.
GUI	Graphical User Interface.
HACC	Heterogeneous Accelerated Compute Clusters.
HBM	High-Bandwidth Memory.
HCGP	Hybrid-Complete Graph and Partial Crossbar.
HD	High Density.
HDL	Hardware Description Language.
HLS	High Level Synthesis.
hls4ml	high-level synthesis for machine learning.
HP	High Performance.

HPC	High-Performance Computing.
I/O	Input/Output.
IaaS	Infrastructure-as-a-Service.
IDE	Integrated Development Environment.
IIC	Inter-Integrated Circuit.
ILA	Integrated Logic Analyzer.
IOP	Input/Output Processor.
IoT	Internet of Things.
IPC	Inter-Process Communication.
ISA	Instruction Set Architecture.
JSON	JavaScript Object Notation.
JTAG	Joint Test Action Group.
MCA	Modular Component Architecture.
MGT	Multi Gigabit Transceiver.
MPB	Main Processing Board.
MPI	Message Passing Interface.
MPSoC	Multiprocessor System on Chip.
NI	Network Interface.
NNNS	Non-uniform node Non-uniform System.
NNUS	Non-uniform node Uniform System.
NoC	Network-on-Chip.

OS	Operating System.
OSFA	Open Source Framework Architecture.
PAR	Place and Route.
PCB	Printed Circuit Board.
PCIe	Peripheral Component Interconnect express.
PGAS	Partitioned Global Address Spaces.
PLD	Programmable Logic Device.
PLL	Phased-Locked Loop.
PMU	Platform Management Unit.
PSD	Pulse Shape Discrimination.
PTK	Parallel Tool Kit.
QP	Quadro Plex.
RAMP	Research Accelerator for Multiple Processors.
RCC	Reconfigurable Computing Cluster.
RDMA	Remote Direct Memory Access.
RM	Resource Manager.
RTC	Real Time Control.
RTL	Register Transfer Level.
SATA	Serial Advanced Technology Attachment.
SBC	Single Board Computer.
SIRCA	Strategic Infrastructure for Reconfigurable Computing Applications.

SNN	Spiking Neural Networks.
SoC	System On Chip.
SoM	System On Module.
SP	Simulation Processor.
SPI	Serial Peripheral Interface.
SPMD	Single-Program Multiple-Data.
SRAM	Static Random Access Memory.
SSD	Solid State Drive.
SSH	Secure Shell.
TDM	Time-Division Multiplexing.
TDMA	Time-Division Multiple Access.
TLM	Tow-level Mesh.
UART	Universal Asynchronous Receiver-Transmitter.
UDMA	Universal Direct Memory Access.
UDP	User Datagram Protocol.
UNNS	Uniform node Non-uniform System.
UNUS	Uniform node Uniform System.
URL	Uniform Resource Locator.
USB	Universal Serial Bus.
XACC	Xilinx Adaptive Compute Clusters.
XSA	Xilinx Support Archive.
ZBT	Zero-Bus Turnaround.

Introduction

Motivation

Supercomputers have been present for almost 60 years. During this period, great inventions were made in hardware and software, pushing computing capabilities further. With each new supercomputer generation, bigger data sets could be processed and more complex algorithms could be implemented, providing more value and further encouraging bigger and faster supercomputers. Nevertheless, it is clear that the current growth trend is suffering from the stagnation of Moore's law, which required doubling the transistor density with each generation. In addition, on the current scale, the energy required to power high-performance computing (HPC) facilities has increased to millions of watts. Cooling facilities and infrastructure costs continue to increase with larger, faster, and more power-hungry chips. At the same time, the dark silicon menace is pervading and the potential of newer chips is eroded by it. In recent years, general-purpose graphical processing units (GPGPUs) and multicore processors have been considered as a way to continue growing in performance. This has shown some promise, but other technologies have shown greater potential.

One of the most critical aspects of traditional computing is memory access that cripples von Neumann architectures. It was shown in a Google [18] research paper that CPUs would suffer from cache miss half of the time. This increases the energy and time budget required for a given computation which needs access to external memory. Considering that when accessing external memory latency and energy increase orders of magnitude compared to accessing internal memory, just by moving processing as close as possible to memory, valuable improvements can be attained. This has not gone unnoticed; in fact, modern trends of in/near memory computation [19] are becoming more common every day.

The most common implementation of in/near memory computation techniques consists in integrating small CPUs inside memory chips. These CPUs have a reduced instruction set architecture (ISA) that enables acceleration of some operations. Nevertheless, one crucial aspect is that the number of operations is reduced and, thus, it is a matter of time before bigger CPUs would be required. This is exactly what major CPU companies are doing, such as AMD, with their EPYC chips consisting of CPU chiplets interconnected with up to 512 GB of high-bandwidth memory (HBM) on the same die [20].

Alternatively, field-programmable gate arrays (FPGAs) offer the possibility of a more flexible approach to in/near memory computing by allowing to instantiate several custom logic circuits to massively parallelize critical tasks, relying on configurable logic blocks and interconnection fabric. FPGAs were initially envisioned as a prototyping platform for application-specific chips (ASICs), while their flexibility made them ideal for low-latency, high-throughput applications with low production volumes. FPGAs evolved into Systems-on-Chip (SoCs or SoC-FPGAs) [21, 9] by including CPUs, digital signal processors (DSPs), embedded memory resources, high-speed transceivers, network-on-chip (NoC), GPUs, and, more recently, neural engines to accelerate deep learning applications. Their capabilities have led to increased interest in specific and general purpose systems [22, 23] with mass adoption yet to occur.

This work focuses on new methodologies and hardware/software architectures for the efficient implementation of SoC-FPGA clusters for supercomputing of interest in both science and engineering. The research plan envisages hardware prototyping, FPGA design, and software development for the creation of the HyperFPGA, a reconfigurable cluster for novel experimental architectures for supercomputing. The hardware development is oriented to define the best computational architecture for the cluster of SoC-FPGA devices and the communication protocol among them. Software development focuses on finding optimal ways to exploit operating systems (OS) to efficiently distribute and execute computations in the cluster, optimizing the involved data transfer times and mechanisms for critical data movement and distributed memory.

The HyperFPGA is based on custom uniform SoC-FPGA boards to build a scalable cluster of cooperative computational units. Each unit of the cluster is configured with a fixed shell to facilitate efficient interaction between the FPGA fabric and its corresponding embedded processor, as well as its interconnections with similar adjacent units. The integrated multicore processor embedded software programming includes a custom Linux operating system, general purpose functions, and routines for communication with the FPGA, standard personal computer for remote control, and user interface.

Given the nature of the HyperFPGA, all design decisions were made to maximize flexibility, from the hardware which consists of carrier boards that host a system-on-module (SoM) for upgradability to the software that relies on open Python functions to allow users to customize the programming flow. The HyperFPGA provides unprecedented plasticity to allow any type of new architecture to be implemented.

As part of the validation process, the HyperFPGA was tested with a common supercomputing problem, the N-Queens problem. By leveraging the massive parallelism of the platform, it was possible to implement a backtracking algorithm with relative ease. Exploiting the depth-first approach of the algorithm, a parallel implementation was carried out to demonstrate the suitability of the HyperFPGA for supercomputing tasks.

Objectives of the Thesis

The objectives of this thesis can be summarized as follows:

1. Perform an analysis of the state-of-the-art.
2. Determine the ideal design characteristics of the cluster.
3. Design and build a modular hardware platform.
4. Build a customizable software framework to implement multiple programming paradigms.
5. Validate the programability and scalability of the platform for supercomputing applications.

Scientific Publications

During the development of this thesis, several journals and conference papers were prepared and published on various topics. In particular, [1,2,3] are directly related to the creation of the HyperFPGA.

In [1], an experimental architecture for SoC-FPGAs is presented. It describes a distributed memory system in which the FPGAs map all resources in a global memory mapped managed locally by entities called local resource agents.

The study of the state-of-the-art is presented in [2]. It delves deep into the architectural decisions of the most relevant FPGA-based clusters by dissecting the clusters into three domains. When this deep inspection is performed, it provides a solid foundation for the HyperFPGA. In [3] the HyperFPGA is presented and described in detail in its three domains. The results of the HyperFPGA experiment are also included in this paper and show the suitability of the HyperFPGA for supercomputing experiments. The software environment presented in [3] offers a user-friendly approach to heterogeneous high-performance reconfigurable computing that shows capabilities for remote education activities, such as the one described in [7].

[1] W. F. Samayoa et al., “An Open-Source Hardware/Software Architecture for Remote Control of SoC-FPGA Based Systems”, in Lecture Notes in Electrical Engineering, Springer International Publishing, 2022, pp. 69–75. doi: 10.1007/978-3-030-95498-7_10.

[2] W. F. Samayoa et al., "A Survey on FPGA-Based Heterogeneous Clusters Architectures," in IEEE Access, vol. 11, pp. 67679-67706, 2023, doi: 10.1109/ACCESS.2023.3288431.

[3] W. F. Samayoa et al. HyperFPGA: An Experimental Testbed for Heterogeneous Supercomputing, 24 August 2023, PREPRINT (Version 1) available at Research Square, doi: 10.21203/rs.3.rs-3278560/v1. Submitted to The Journal of Supercomputing, Springer.

The development of the HyperFPGA was only possible due to valuable research carried out in the context of SoC-FPGA systems. Such research includes studies on asynchronous cell automata networks [4], which are a great fit to experiment with the HyperFPGA given how they benefit from scalable hardware when studying rules beyond 5 inputs. Furthermore, communication between FPGAs was explored in [5, 6] by implementing a time-division multiplexing method in a parallel control system of a low-latency network for precise timing synchronization.

[4] A. Cicuttin et al. , “Physical implementation of asynchronous cellular automata networks: mathematical models and preliminary experimental results”, Nonlinear Dyn., Jul. 2021, doi: 10.1007/s11071-021-06754-z.

[5] L. G. García et al., “High Voltage Isolated Bidirectional Network Interface for SoC-FPGA Based Devices”, in Lecture Notes in Electrical Engineering, Springer International Publishing, 2021, pp. 280–285. doi: 10.1007/978-3-030-66729-0_34.

[6] L. G. G. Ordóñez et al., “Multichannel Time Synchronization Based on PTP through a High Voltage Isolation Buffer Network Interface for Thick-GEM Detectors”, Instruments, vol. 6, no. 1, p. 11, Feb. 2022, doi: 10.3390/instruments6010011.

[7] M. L. Crespo et al., “Remote Laboratory for E-Learning of Systems on Chip and Their Applications to Nuclear and Scientific Instrumentation”, Electronics, vol. 10, no. 18, p. 2191, Sep. 2021, doi: 10.3390/electronics10182191.

Other publications:

In the course of this Ph.D. multiple collaborations were established producing publications related to high-energy physics in the context of the COMPASS and AMBER experiments at CERN [8, 10 ,11 ,12, 13]. Similarly, a collaboration in the context of X-ray spectroscopy produced a publication on data analysis [9].

[8] G. D. Alexeev et al., “Probing transversity by measuring λ polarisation in SIDIS”, Phys. Lett. Sect. B Nucl. Elem. Part. High-Energy Phys., vol. 824, 2022, doi: 10.1016/j.physletb.2021.136834.

[9] K. S. Mannatunga et al., “Data Analysis and Filter Optimization for Pulse-Amplitude Measurement: A Case Study on High-Resolution X-ray Spectroscopy”, Sensors, vol. 22, no. 13, p. 4776, Jun. 2022, doi: 10.3390/s22134776.

[10] J. Agarwala et al., “MPGD-based photon detectors for the upgrade of COMPASS RICH-1 and beyond”, J. Instrum., vol. 15, no. 9, 2020, doi: 10.1088/1748-0221/15/09/C09063.

[11] S. Carrato et al., “A scalable High Voltage Power Supply System with system on chip control for Micro Pattern Gaseous Detectors”, Nucl. Instrum. Methods Phys. Res. Sect. Accel. Spectrometers Detect. Assoc. Equip., vol. 963, p. 163763, May 2020, doi: 10.1016/j.nima.2020.163763.

[12] A. Bressan et al., “The high voltage system of the novel MPGD-based photon detectors of COMPASS RICH-1 and its development towards a scalable High Voltage Power Supply System for MPGDs”, Nucl. Instrum. Methods Phys. Res. Sect. Accel. Spectrometers Detect. Assoc. Equip., vol. 1056, p. 168558, 2023, ISSN 0168-9002, doi: 10.1016/j.nima.2023.168558.

[13] W. F. Samayoa et al., (2023). Diagnostic Analytics for Pixelated Particle Detectors: A Case Study. In: Berta, R., De Gloria, A. (eds) Applications in Electronics Pervading Industry, Environment and Society. ApplePies 2022. Lecture Notes in Electrical Engineering, vol 1036. Springer, Cham. doi: 10.1007/978-3-031-30333-3_28

Thesis Outline

The remainder of the thesis is organized as follows. In Chapter 1, the state-of-the-art is presented and described in detail. From the state-of-the-art study, a specific discussion leading to platform design is elaborated in Chapter 2. The hardware of the HyperFPGA, comprising the printed circuit board (PCB) components, interfaces, and power domains, is discussed in Chapter 3. The software tools and the development environment are presented in Chapter 4. In Chapter 5, an application problem is presented along with the results obtained with the HyperFPGA. Finally, in chapter 6, conclusions are drawn, highlighting the main contributions and directions for future research.

Chapter 1

State of the art

The notion of reconfigurable hardware has been present since 1984, when Altera introduced the first programmable logic device (PLD) to the industry [24]. Then, in 1985 Ross Freeman and Bernard Vonderschmitt patented the first commercially viable FPGA [25]. Due to production costs, compared to ASICs, FPGAs are traditionally used in applications with low production volumes that require high throughput and low latency.

FPGAs are electronic devices that consist of many configurable logic blocks composed of look-up tables, flip-flops, input/output (I/O) blocks, and interconnection fabric. Being reconfigurable hardware, in FPGAs algorithms are implemented as custom digital circuits, heavily contrasting from the software approach used for CPUs. The initial step typically consists of describing the algorithm using a Hardware Description Language (HDL), such as VHDL or Verilog. The HDL description is then synthesized into a netlist that is mapped onto the FPGA's logic elements and interconnections required to implement an equivalent digital design. The final implementation in the FPGA is performed using vendor-specific tools such as Vivado [26], Vitis [27], Quartus [28], and Libero [29]. Once the mapping and routing process is complete, the design is compiled into a bitstream file that can be loaded onto the FPGA to configure its logic elements and interconnections to create a circuit corresponding to the algorithm. Proprietary FPGA vendor tools have dominated the field, but in recent years open source FPGA tools, such as Yosys [30], F4PGA [31], and RapidSilicon [32], have surfaced providing an alternative for research and power users.

As a reconfigurable device, FPGA offers the advantage of continuous improvement in hardware and software. Being able to change architecture offers great freedom when developing complex systems. Furthermore, FPGAs have been shown to consume considerably less power than CPUs and GPUs [7], leading to reduced cooling and energy costs.

By studying computing problems, a classification based on repeating algorithmic patterns was proposed in 2004 [33] and later refined [17] to 13 algorithmic encapsulations, named the 13 dwarfs, shown in Table 1.1. Theoretically, each dwarf can be assigned to a specific computing architecture [16, 34], CPU, GPU, or FPGA, which encourages heterogeneity in high-performance computing platforms. Likewise, this inspired the creation of benchmarks for heterogeneous systems such as DwarfBench [35], Rodinia [36], and OpenDwarfs [37].

Table 1.1: The 13 dwarfs [17] with application examples, where each represents an algorithmic method that encapsulates the patterns of communication and computation.

	Dwarf	**Example applications**
1	Dense matrix	Linear algebra (e.g., Cholesky decomposition)
2	Sparse matrix	Linear algebra (e.g., machine learning)
3	Spectral	FFT-based methods
4	N-body	Particle-particle interactions, molecular dynamics
5	Structured grid	Fluid dynamics, meteorology
6	Unstructured grid	Adaptive mesh FEM
7	MapReduce	Monte Carlo integration, distributed pattern-based searching
8	Combinational	Logic gates (e.g., N-Queens)
9	Graph traversal	Searching, selection
10	Dynamic programming	Tower of Hanoi problem
11	Backtrack/global optimization	Branch-and-bound
12	Graphical models	Probabilistic networks
13	Finite state machine	Transistor-transistor logic counter

Several implementations of heterogeneous HPC systems housing FPGAs can be named, such as *Project Catapult* at Microsoft [38], *Alibaba FaaS* (FPGA as a Service) [39], *Amazon EC2 F1* instances [40], and *ARUZ* cluster at Lodz University [41]. At *CERN*, the massive adoption of FPGAs for online data processing has motivated the development and adoption of specific tools to aid the development of applications based on FPGAs, such as *hls4ml* [42] (high-level synthesis for machine learning). This tool, along with many others [43, 44, 45, 46, 47], allows for a higher level of abstraction, thereby significantly reducing implementation errors and development time. The preference for FPGAs is due to their reconfigurability, which allows extreme hardware specialization when needed. In addition, the fact that FPGAs offer a wide array of input-output ports makes them ideal for stream computation and for creating pipelined systems that can maintain high throughput with low latency.

With such a significant number of parameters to optimize when designing and implementing FPGA clusters, they have mainly been developed with an application-specific mindset. The heterogeneous aspects of most applications make compromising flexibility in different domains (hardware, network, or software) reasonable. In the following sections, some of the most relevant FPGA-based clusters are introduced with their contributions. Each cluster is broken down to its smallest functional units from which the systems scale, called computational units (CUs). Likewise, each CU is composed of one or many diverse computing elements (CEs), namely CPUs, GPUs, and FPGAs, which actually perform the processing.

1.1 Application fields

Given that FPGAs serve specific purposes, they have found a place in some computing fields. One of the first applications field of FPGAs was the emulation of integrated chips, particularly multicore chips. From this field, the potential of FPGAs was demonstrated leading to experimental implementations in science computing and other fields requiring high-throughput of online data processing. The specific requirements of each application brought to light numerous advantages of reconfigurable computing architectures, while also showing the difficulty of working with these technologies.

1.1.1 Emulation of multicore processors

Developing multicore integrated chips is a long and expensive process that involves several stages of experimentation, validation, and integration. There are software tools that help simulate architectures for easy parameter tuning, with the major drawback of speed. In this particular aspect, FPGA prototyping allows faster execution times and benefits from insights from real hardware. It is not uncommon for a complete platform to exceed the logic available in a single FPGA, pushing for a cluster of FPGAs.

This was the case since 1997 when one of the first FPGA clusters was used to emulate the *RAW* architecture [48]. The *RAW* cluster consisted of 5 boards or CUs, each with 64 FPGAs, totaling 320 FPGAs. Its results showed orders of magnitude speed-up compared to contemporaneous scalable processors with the disadvantages of reduced flexibility, high cost, and high implementation complexity, which hindered their adoption in other research applications.

In 2006, the *FAST* [10] cluster was presented to bring hardware back into the research cycle to address the disadvantages of *RAW*. FAST combined dedicated microprocessor chips and static random access memories (SRAM) with FPGAs into a heterogeneous hybrid solution to simulate chip multicore architectures. The vision was to reduce hardware costs and ease development, both for programming and portability. Each *FAST* CU consisted of 8 processors, 10 Xilinx Virtex FPGAs, and 4 tiles interconnected with memory. The 2 processors in each tile acted as the CPU and floating processing unit, respectively, and the 2 FPGAs acted as the level-one memory controller and co-processor.

A central hub, made up of 2 FPGAs, was used to manage shared resources and orchestrate communication between tiles, allowing access to off-the-board devices through external I/Os. Additionally, the expansion connector available to the FPGA hub allows multiple *FAST* CUs to be connected. The CU implementation is illustrated in Figure 1.1.

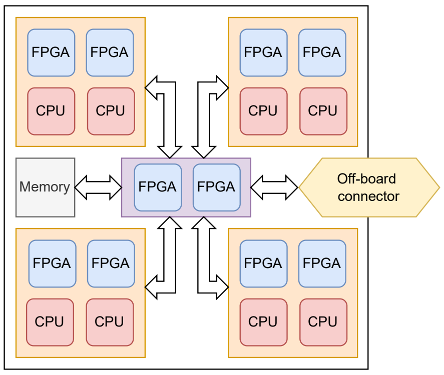


Figure 1.1: FAST [10] computational unit (CU) with the computing tiles in orange and the FPGA hub in purple.

A custom software stack was specifically developed for *FAST*. It included several modules and pre-defined interfaces for functionality and benchmarking. An operating system was developed to manage control tasks, such as programming and configuration. Portability was demonstrated by implementing several architectures; however, scalability and costs remained open to discussion.

Similar to *FAST*, the *RAPTOR* cluster was presented as a baseboard that hosts up to 4 daughter cards based on complex programmable logic devices (CPLD) [49]. In 2010, a second version was presented using FPGAs and a renewed architecture [11]. This new version consisted of a *RAPTOR-Xpress* baseboard (CU) that provides two buses for Gigabit Ethernet (GbE), universal serial bus (USB) 2.0, and peripheral component interconnect express (PCIe) 2.0 x8 for the host connection to configure and manage up to 4 DB-V5 (daughter board version 5).

Figure 1.2 shows the *RAPTOR-Xpress* baseboard with 4 DBs that directly interface with their neighbors in a ring topology. Each has a Xilinx Virtex-5 FPGA with up to 4 GB of DDR3 memory and a dedicated FPGA as a PCIe interface. Multiple baseboards can be connected via 4 high-speed connectors, each consisting of 21 full-duplex serial lanes, allowing resource scaling beyond the 4 DB on board. The baseboards can also be interfaced with the host via dedicated FPGAs in Nallatech front-side bus acceleration modules [50], providing an additional 8.5 Gb/s for writing and 5.6 Gb/s for reading.

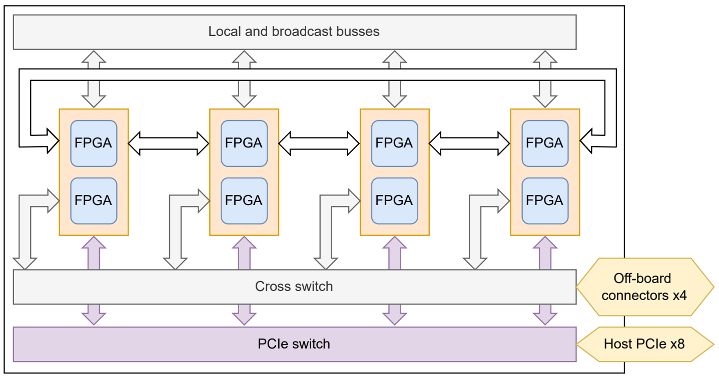


Figure 1.2: Simplified diagram of the RAPTOR-Xpress board [11] or computational unit (CU) with the daughter boards in orange.

The *RAPTOR* project also includes a custom software development environment that includes *RAPTORLIB*, *RAPBTORAPI*, and *RAPTORGUI* tools, which aid developers by providing hardware-supported protocols, remote access, and a graphical user interface to facilitate testing. The design flow offers tools for design partitioning, which is a manual process assisted by a graphical integrated development environment (IDE) and standard synthesis tools developed in *vMAGIC* [51].

Convinced by the need for cheaper and smaller hardware, the *Formic* cluster [52] based on the *Formic* board [53] was presented in 2014. The *Formic* board acts as the building block for a larger system, with a maximum size of 4096 boards. All boards consist of an FPGA, SRAM, 1 GB of double data rate (DDR) RAM, a power supply, buffered joint test action group (JTAG) connectors, and configuration memory, making them independent and perfectly symmetric. Eight multi-gigabit transceivers (MGT) at a maximum speed of 3 Gb/s are available for interconnection on 8 serial advanced technology attachment (SATA) connectors. Inside each board, a full NoC with a 22 port crossbar switch interfaces the configured blocks with MGT links and allows developers to scale the designs. Access to local and remote memories is done using the Remote Direct Memory Access (RDMA) protocol [54]. The first application consisted of a 512-core cluster [55] based on 8 custom MicroBlaze [56] processors per module.

Moreover, the industry has produced exciting developments in multicore emulation. In an attempt to reduce the time to market for new ICs, *Cadence* [57] and *Siemens* [58], among others, developed solutions for the prototyping of ICs. Unfortunately, there is little accessible information regarding the architecture of most implementations, and the high costs make them uncommon in academia, with some exceptions, such as the *Pico Computing* board (now *Micron*) used for image processing [59] and the *DINI* (now *Synopsys*) board FPGA board used for online video processing [60].

From the described works, it can be seen that there is a trend to reduce the complexity of CUs, as shown in Figure 1.3. In this field, costs tend to be the leading factor, making granularity a desirable characteristic. With smaller CUs, it is possible to reduce the implementation costs, depending on the requirements of the chip to emulate. Smaller CUs also make it easier for clusters to scale, maintain, and upgrade.

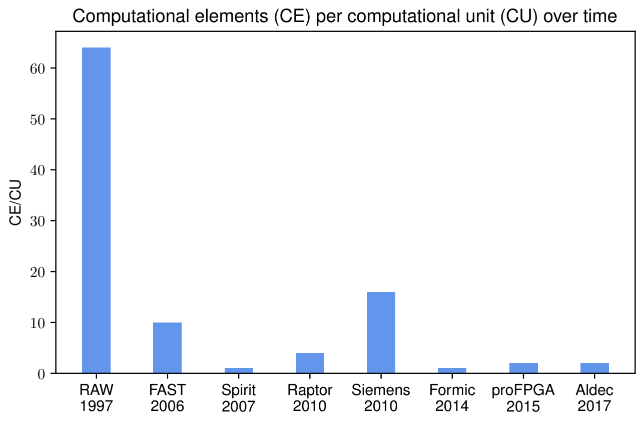


Figure 1.3: Clusters targeting multicore emulation have shown a trend of reducing the complexity and increasing the granularity of computational units (CUs) to favor production costs and scalability.

Table 1.2 shows a summary of the most relevant clusters in the field of manycore emulation along with their contributions and performance improvements.

Table 1.2: multicore emulation clusters' contributions and reported performance improvement.

Work	Contribution	Performance
RAW [48]	First academic FPGA cluster for the emulation of multicore systems.	10 to 1k times more performant than commercial Sparc 20/71.
FAST [10]	First hybrid platform for multicore research using dedicated FPGAs, SRAMs, and microprocessors.	Over 2x more performant than any previous emulation platform.
RAPTOR [11]	Scalable MPSoC emulator with multiplexed PCIe access from the host.	
Formic [53]	Cost-efficient scalable cluster based on a minimal independent building block.	50k times faster than that of the software simulation.

1.1.2 Scientific computing

The complexity of scientific computing problems has always pushed technology to its limit, making computer clusters a basic requirement. Regardless of whether complex algorithms process huge amounts of data or massive system simulations, reconfigurable computing provides the level of customization often required by these problems. Since 1991, programmable hardware was already part of custom supercomputers for specific problems like in *RTN* [61], *RASA* in 1994 [62], and later in *SUE* 2001 [63].

The first massive cluster was created in 2006. *Janus* [64] was a massively parallel modular cluster for the simulation of specific theoretical problems in physics developed by a large collaboration of European institutions [65].

The core of *Janus* comprised an array of 4 by 4 FPGA-based simulation processors (SP) that were connected with their nearest neighbors. Another processing unit called an Input/Output processor (IOP), acted as a crossbar and was in charge of managing communications between FPGAs and the host.

A two-layer software stack was created to help developers build applications. The firmware layer consisted of a fixed part targeting the IOPs, which included a stream router and dedicated

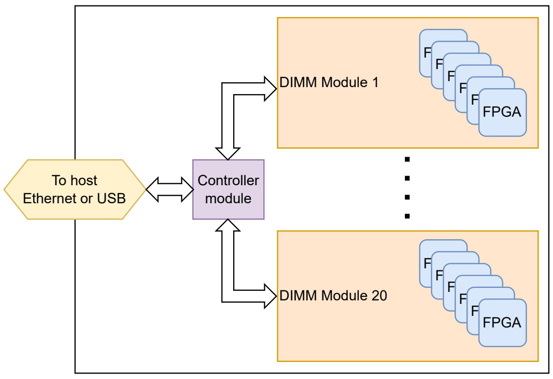


Figure 1.4: *COPACOBANA* [12] computational unit (CU) with dual inline memory module (DIMM) modules in orange, each with 6 FPGAs, and the controller module in purple.

devices to communicate, manage, and program the SPs. The second layer, the *Janus* Operating System (*JOS*), consisted of the programs running on the host PCs, including a set of libraries (*JOSlib*) to manage IOP devices, Unix socket application program interfaces (APIs) to integrate high-level applications and new SP modules, and an interactive shell (*JOSH*) for debugging and testing.

In the worst case, *Janus* performed just 2.4 times faster than conventional PCs. However, *Janus* was limited by its performance and limited memory for some applications [66].

In parallel, great interest was shown in the field of cryptanalysis with the development of the *COPACOBANA* FPGA cluster [12] in 2006. Figure 1.4 shows the *COPACOBANA* cluster which was built over a CU that held up to 20 dual inline memory modules each with 6 Xilinx Spartan-3 FPGAs directly connected to a 64-bit data bus and a 16-bit control bus. A controller module allowed the host PC to interact via USB or Ethernet through a software library that provided the necessary functions for the PC to program, store, and read the status of the cluster as a whole or as individual FPGAs. This made it possible to scale resources by attaching another CU to the host PC. Its capabilities were demonstrated by testing several encryption algorithms, which resulted in it outperforming conventional computers by orders of magnitude [67].

The positive result of this project motivated the creation of a hybrid FPGA-GPU cluster [68] based on commercial off-the-shelf (COTS) components in 2010. The *Cuteforce* [69] system implemented 15 CUs, 14 with Xilinx Virtex FPGAs, and the last with an NVIDIA GPU interconnected through a CPU on a CU via Infiniband. The results were not as expected, in part because of complications in FPGA implementation.

The same approach was later used in 2010 by *Tse, et al.* [70] who focused on Monte Carlo simulations. However, instead of using one CE per CU, a single CU was used to host 2 CPUs, an NVIDIA GPU, and a Xilinx FPGA, which was further supported by a comprehensive analysis of the performance and energy. The network remained practically unchanged from *Cuteforce*, where the CPUs are the main communication CEs and relegate GPUs and FPGAs to an accelerator position. To further demonstrate the scalability of this strategy, *Superdragon* [71] was created to accelerate single-particle cryoelectron 3D microscopy. Similarly to its predecessors, *Superdragon* implemented a mixed parallel approach in which the load is distributed in the CPUs using task-parallelism to later offload particular tasks on GPUs and FPGA using data-parallelism. A set of pre-defined blocks are made available to the user in a dataflow module (DFM) with runtime reconfigurability of the data path.

Bluehive [72] also sought to distance itself from custom PCBs by embracing commodity boards to build a custom FPGA cluster for scientific simulations and multicore emulation [73] requiring high-bandwidth and low-latency communication. These challenges were overcome with the development of a 64-node FPGA cluster based on Terasic DE4 boards that host an Altera Stratix IV FPGA, an 8xPCIe connector, and a DIMM with 4 GB of RAM and interfaced through a custom interconnect called BlueLink [74] with four 8U rack boxes, each with 16 boards. The boards in the boxes were interconnected through a PCIe to the eSATA board. A small Linux computer allowed remote programming using a USB-to-JTAG converter and a DE2 board as a JTAG fan-out to parallelize the configuration.

The *Bluehive* development environment was supported by Quartus. Blocks were provided to developers, routers for inter-FPGA communication, functional blocks (FBs), and high-speed serial link controllers [74], all developed on Bluespec SystemVerilog [75]. Its effectiveness was demonstrated with a simulation of 64k spiking neurons.

In 2014, *Janus* received an important upgrade [76], which significantly improved its performance. The architecture remained mostly the same, with the largest change in the adoption of newer FPGAs with 8 GB of RAM and MGTs instead of ordinary I/Os for interconnection.

Janus 2 and *Bluehive* were successful in addressing the memory problem, but as the applications scale, larger clusters were needed. This was the case for *ARUZ* [41], an application-specific cluster formed by approximately 26,000 FPGAs distributed over 20 panels, each consisting of

12 rows, which in turn contained 12 CU. The CUs are made up of eight slave FPGAs that constitute the resources and a central master SoC-FPGA that manages operations. The addition of the SoC-FPGA is motivated by the higher abstraction level provided by the ARM processor for slow-control tasks. In addition, each CU is interfaced with a concentrator board (CB) that feeds the state of the simulations to a host that controls the entire process.

Global communication is based on Gigabit Ethernet and allows data exchange between SoC-FPGAs to configure its 8 FPGAs. All nodes are connected in a daisy chain, and only one board is connected to an external switch. A custom protocol for data transfer was developed consisting of a small packet of no more than 256 bytes, with a constant overhead of 11 bytes.

ARUZ designers developed their own methodology [77], as there are no standard solutions available. Taking into account the multitude of combinations of mechanisms for programming and controlling *ARUZ*, a high level of flexibility is required. *VPP* [78] was selected for code pre-processing and parameterization. DLLDesigner was developed to generate VHDL code to interconnect as many FBs as required. All of these tools allow for the implementation of highly optimized architectures for molecular simulations.

FPGAs have also found a place in neuromorphic computing, as demonstrated by *Bluehive*. Spiking neural networks (SNN) require many densely interconnected elements. A substantial level of parallelism is suitable for hardware acceleration; however, the challenge is scalability. This was specifically addressed by *Astrobyte* [79] using a fully scalable NoC-based FPGA cluster with functional verification and real-time monitoring. However, more specialized platforms presented better results at higher costs. This is the case for *BiCoSS* [80], a 35 system-on-module cluster, each with a Cyclone IV FPGA and 2 SDRAMs capable of simulating 4 million spiking neurons in real-time.

Another relevant application in the scientific context is real-time control (RTC) systems of adaptive optics (AO) instruments. This is the main focus of the *Green Flash* project [81] that aims to develop energy-efficient real-time HPC accelerators and smart interconnects, based on GPUs and FPGAs [82]. The RTC modules have a standard CPU server that hosts an NVIDIA GPU, Intel CPU, and Intel Arria 10 FPGA. The FPGAs are hosted on a custom mainboard called $\mu XComp$ which includes 2 GB of RAM onboard, PCIe 3, an FMC connector, Ethernet, 4 QSFP, and other valuable resources.

In this heterogeneous system, communication between GPUs is performed by a Smart Interconnect system implemented on FPGAs. The SI uses the UDP protocol, which is implemented in the FPGA fabric alongside the device protocol handlers and dedicated direct memory access (DMA) engines. This is configured with the QuickPlay FPGA framework, which extends its capabilities by using abstraction models and board support packages (BSPs) for portability. This architecture allows us to pipe-line several GPUs and FPGAs. A similar approach can be seen in *Spinnaker* [83] and *BrainscaleS* [84] supercomputers, which implement dedicated ASICs interconnected by FPGAs for neuromorphic computing, and *MDGRAPE* [85] for modular dynamics simulations.

Table 1.3 resumes the most relevant scientific computing clusters and shows their contributions with the impact in power consumption and performance.

Table 1.3: Scientific computing clusters' contributions, reported power and performance gains.

Work	**Contribution**	**Power**	**Performance**
Janus [64]	Fine-grained memory structures for statistical physics simulations on a reconfigurable cluster, evidence of memory as the performance bottleneck.	9x more efficient than an Intel Core 2 Duo cluster.	1k times faster than Intel Core 2 Duo cluster.
COPACOBANA [12]	Cost-effective FPGA cluster for cryptanalysis.	8x more efficient than a PC cluster.	More than 650x faster than an equivalent cost PC (Pentium-M) for exhaustive DES key search.
Cuteforce [69]	Efficient framework for FPGA-GPU collaborative cluster programming.		7.9M keys per second for PDF encryption brute-force attack.

Continued on next page

Table 1.3: *Continued from previous page*

Work	**Contribution**	**Power**	**Performance**
Tse, et al. [70]	Research on efficiency of collaborative clusters of CPUs, GPUs, and FPGAs with dynamic scheduling showing that FPGA and GPU collaboration offers the best energy-performance trade-off.	19.6x more efficient.	44x faster with FPGA-GPU nodes over 2 CPU nodes (AMD Phenom 9650 quad-core).
Bluehive [72]	Usage of commercially available boards as building blocks to simulate millions of neurons in real-time.		162x faster than software simulation.
Janus 2[76]	Janus update with more memory, larger FPGAs, and tightly coupled to host computers.	12x more efficient than Xeon-Phi.	26x faster than a Xeon-Phi.
Superdragon [71]	CPU, GPU, and FPGA cluster for 3D reconstruction of cryoelectron microscopy mixing parallel patterns.	GPU and FPGA improved efficiency by 7.2x and 14.2x, respectively compared to a Multicore CPU.	GPU and FPGA speed up of 8.4x and 2.25x, respectively, compared to a multicore CPU.
ARUZ [41]	Biggest FPGA cluster to date based on the Dynamic Lattice Liquid model with low latency communication.	1.6x more efficient than a 6-core CPU for simulating 1.5M molecules.	1600x faster than a 6-core CPU.

Continued on next page

Table 1.3: *Continued from previous page*

Work	**Contribution**	**Power**	**Performance**
Greenflash [82]	Heterogeneous cluster for critical real-time application.	FPGA and GPU parts show double the energy efficiency compared to Xeon Phi CPUs.	GPU and FPGA parts show an improvement of 2.6x and 3.5x, respectively, over Xeon Phi CPUs.
Astrobyte [79]	FPGA cluster for simulating spiking neuron networks with NoC routing mechanisms.		188 times faster compared to Matlab implementation.
BiCoSS[80]	Biologically inspired multi-FPGA cluster for neuromorphic computing.	2.8k times more efficient than an NVIDIA GTX 280 GPU platform.	Computational efficiency of 540k times more than traditional CPU-based serial solutions.

1.1.3 FPGAs in data centers

The positive results obtained by FPGAs attracted great interest outside of the scientific community, specifically in the data center (DC) context, where computing tasks can quickly overwhelm CPUs. DC workloads demand reduced power consumption, latency, and cost while maximizing computing power and flexibility.

Catapult [38] is a successful example of the inclusion of FPGAs in high-reliability commodity DC. FPGAs were specifically selected given that the flexibility of reconfigurable hardware helps tackle the 2 main requests in DCs. First, the desire for homogeneity greatly facilitates the installation, maintenance, and deployment of services. Second, there is a need for flexibility, considering that such services evolve rapidly, making fixed hardware impractical.

A custom half-width unit motherboard was developed to host 2 high-end CPUs and the daughter FPGA card, which consisted of a Stratix V D5 FPGA with 8 GB of DRAM and acted as the CU. Two 12-core Sandy Bridge processors with 64 GB of RAM, 2 SSDs, and 4 HDDs complete the resources present on the motherboard. The FPGA and host CPUs communicate via PCIe, and high-speed transceivers are used in the inter-FPGA network. A two-dimensional 6x8 node torus was selected for the network configuration in each rack. For the final system, 34 of these racks were used for a total of 1632 nodes.

To evaluate the performance of *Catapult*, a significant portion of the ranking stack of Bing was offloaded to each rack. To guarantee the reliability of the system, the following services were implemented: 2-bit error detection and 1-bit error correction on top of the CRC in the DRAM and high-speed network. For user productivity and reusability, the FPGA space was split into 2 parts. A shell that hosts hardware controllers, an inter-FPGA network stack, a status notifier, and a single-event upset logic to reduce system errors that consume 23% of the FPGA resources, and a role part where the computing logic lies. In addition, a mapping manager and health monitor continuously scanned each node in the network. In case of failure, the faulty node is immediately reconfigured. If the issue persists, the node is flagged for manual intervention, and the mapping manager automatically relocates the services to the available resources.

With custom hardware and communication protocol, *Catapult* achieved an improvement of 95% in throughput in a production search infrastructure compared to a software-only solution. Furthermore, the inclusion of the FPGA increased power consumption by only 10%, and the added cost of ownership did not exceed the limit of 30%. These results show the significant advantage that FPGAs can offer in terms of throughput and power consumption.

With the success of *Catapult* [38], it was only a matter of time before FPGAs were made available for cloud computing tasks, which is exactly what *IBM cloudFPGA* [86] did. Virtualizing the user space makes FPGAs in an Infrastructure-as-a-Service (IaaS) environment feasible for education, research, and testing.

In the architecture presented, the FPGAs are standalone nodes in the cluster directly interfaced to the DC via PCIe, unlike the approach of *Amazon* [40], *Alibaba* [39] and *IBM Supervessel*[87] which bind the FPGAs to the host CPUs. Under this approach, a daughter card consisting of an FPGA and abundant RAM was developed. By creating a custom carrier board, 64 daughter cards can be accommodated in a single 2U rack chassis [88]. To achieve the desired homogeneity within the DC, FPGAs have been provided with a soft network interface chip, with the advantage of loading only the required services.

The multi-FPGA fabric formed by multiple prototypes of a network-attached FPGA was evaluated with a text-analytics application. The results, compared to a software implementation and an implementation accelerated with PCIe-attached FPGAs, show that the network-attached FPGAs improved in latency and throughput. Additionally, network performance was compared with bare metal servers, virtual machines, and containers [89], with results showing orders of magnitude better for the FPGA prototype. To further improve the usability of the platform, continuous development has been carried out to integrate MPI into the system [90, 91] and OpenMP implementations [92, 93]. An in-depth study of FPGA cloud computing architectures is available in [94, 95].

A summary of the main heterogeneous cluster in the domain of data center applications is presented in Table 1.4 with their contributions, power consumption metrics, and performance gains.

Table 1.4: Data center FPGA clusters' contributions reported power and performance improvement.

Work	**Contribution**	**Power**	**Performance**
Catapult [38]	An FPGA-based cluster to accelerate the Bing web search engine ranking service.	Peak power consumption of 287 GFLOPS/W when running large deep neural network models with high device utilization.	Twice the improvement in search throughput and a 29% reduction in the latency delay to process the search.

Continued on next page

Table 1.4: *Continued from previous page*

Work	**Contribution**	**Power**	**Performance**
IBM cloudFPGA[86]	Disaggregated FPGAs to increase computing density in racks for multi-tenant cloud system.	10x more efficient than the MareNostrum 4 supercomputer	Peak performance of 344M force interaction calculations per second for the N-body problem.

1.1.4 General-purpose clusters

Over-specialized systems tend to constrain the potential of reconfigurable hardware in favor of optimizing performance or costs for a particular application. However, general-purpose clusters are addressed by a larger group of projects that seek to change the current programming paradigm. These clusters, rather than being general-purpose in the broad sense of the word, serve as experimental platforms to test solutions to all heterogeneous supercomputing challenges, ranging from network to user experience.

One of the first projects was the Reconfigurable Computing Cluster (RCC) [96] in the early 2000s. It was a multi-institution investigation project that explored the use of FPGAs to build cost-effective petascale computers, with its main contribution being the introduction of microbenchmarks for software, network performance, memory bandwidth, and power consumption. To evaluate each test *Spirit*, a cluster was built consisting of 64 FPGA nodes. Each node had a Virtex 4 FPGA with 2 Gigabit Ethernet ports, 8 DIMM slots for onboard RAM, and 8 MGTs for board-to-board interconnection [97] using the *Aurora* protocol [98].

For internode communication, a configurable network layer core was developed as part of an Adaptable Computing Cluster project [99]. It consists of a network switch implemented in the FPGA that acts as a concentrator for the router.

Considering that the head node is a workstation, a message-passing interface (MPI) approach offered the flexibility that the cross-development environment required. A custom compiler based on GNU GCC was built to support MPI and its modular component architecture (MCA) [100], which was adapted to support the high-speed network. A software infrastructure based on a Linux system allowed users to access, manage, and configure all nodes in the cluster via SSH [101].

Similarly to the *RCC* project, the FPGA High-Performance Computing Alliance (FHPCA [102]) was established in 2005 with the *Maxwell* supercomputer [103]. The *Maxwell* CUs were built on a standard IBM BladeCenter chassis, in which an Intel Xeon and 2 FPGAs were interfaced via PCI-X. Additionally, an FPGA-dedicated network is available via MGTs without routing logic, given the nearest-neighbor scheme. By supporting standard parallel computing software, structures, and interfaces, it sought to disrupt the HPC space without causing significant friction.

To facilitate the development of applications targeting *Maxwell*, the Parallel Toolkit (PTK) [104] was developed. It included a set of practices and infrastructure to solve issues such as associating tasks with FPGA resources, segmenting the application into bitstreams, and managing code dependency. PTK provided a set of libraries where common standard interfaces, data structures, and components were defined.

Likewise, *Cube* was created to explore the scalability of a cost-effective massive FPGA experimental cluster for real-world applications. It consisted of 8 boards that host a matrix of 8 by 8 Xilinx FPGAs [13] that form a cluster of 512 FPGAs, as shown in Figure 1.5. It features a single configuration of multiple data programming paradigms that allowed all FPGAs to be configured with the same bitstream in a matter of seconds. The FPGAs were interconnected in a systolic array that reached up to 3.2 Tb/s inter-FPGA bandwidth, offering significant advantages as it simplified the programming model and greatly relaxed the requirements of the PCB layout.

Simultaneously, *Quadro Plex (QP)* [105], a hybrid cluster was introduced. It was composed of 16 nodes, each consisting of one AMD CPU, 8 GB of RAM, 4 NVIDIA Quadro GPUs, and one Xilinx Virtex 4 Nallatech FPGA accelerator. The nodes were interconnected using Ethernet and Infiniband. Cluster communication was managed using the OpenFabrics Enterprise Distribution software stack. The complete system occupied four 42U racks, consumed 18 kW, and had a theoretical performance of 23 TFLOPS. CUDA was used for GPU development, and the FPGA workflow was entirely dependent on the Xilinx ISE design suite [106].

Several applications were developed, showing that there were substantial difficulties in taking advantage of the entire system. Applications would only use a combination of CPUs and GPUs

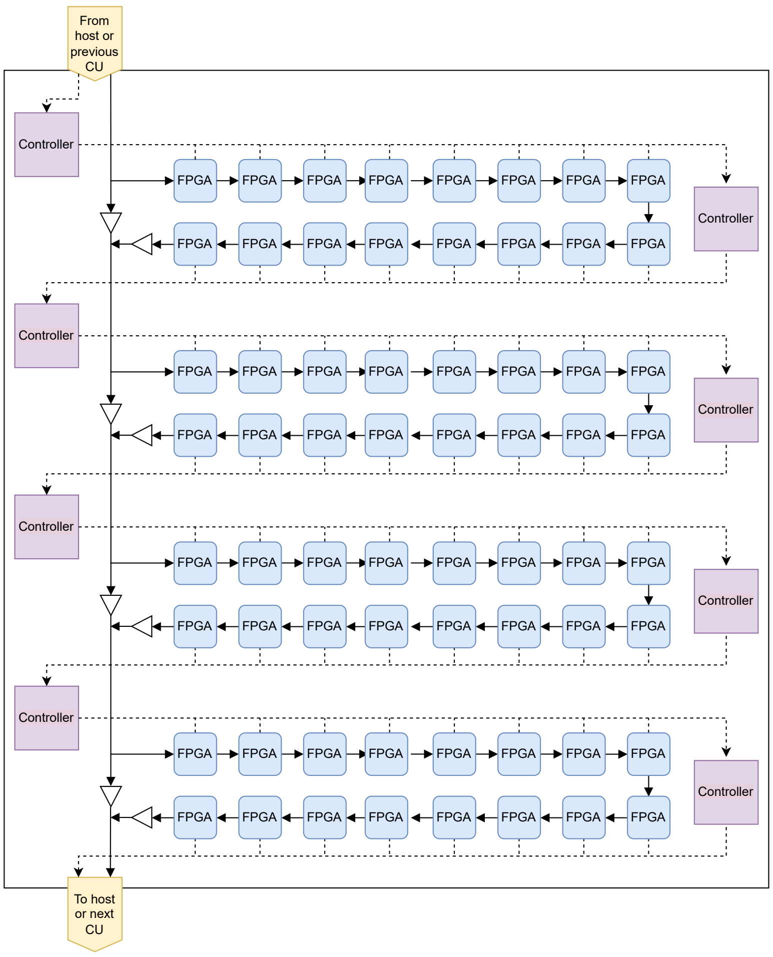


Figure 1.5: *Cube* [13] computational unit (CU) showing the configuration controllers in purple. Dotted lines show the control and configuration bus, and solid lines show the data path.

or CPUs and FPGAs. A framework was developed to facilitate the porting of applications and provide a compatibility layer for different accelerator workflows, called Phoenix [107].

In the same spirit, *Axel* [15] was built. It consisted of 16 nodes, each node with an AMD CPU, an NVIDIA Tesla GPU, and a Xilinx Virtex 5 FPGA occupying a 4U full-scale rack. All CEs were connected to a common PCIe bus for intranode communication and between nodes in a Gigabit Ethernet network. Considering the high latency and non-deterministic nature of Ethernet, a parallel network was also available through the 4 MGT of the FPGA.

The cluster was remotely managed from the central node using the *Torque* [108] resource manager and the *Maui* [109] scheduler. A custom resource manager (RM) was responsible for the management of GPUs and FPGAs. For this to be feasible, all *Axel* programs needed to allocate part of the resources in the CEs to interface with the RM runtime API. Using an inter-process communication (IPC) message queue framework, CEs communicated their state to the head node. The central node collected information from all nodes with the help of the RM and prepared a script to submit the jobs to Torque. Communication between tasks was performed through MPI using gigabit Ethernet.

To implement an application in *Axel*, users provide a data flow graph and a hardware abstraction model. A MapReduce framework then rewrites the application for partitioning the analysis into tasks. These tasks are assigned to the corresponding CEs on the basis of the targeted attributes.

Axel also introduced an architecture classification for heterogeneous systems based on uniformity. Following this classification, *Axel* is a Non-Uniform Node Uniform System (NNUS) architecture. This means that all nodes are equal but built with different CEs. The advantage of this architecture is that the single-program multiple-data (SPMD) programming paradigm can be implemented throughout the system. *Axel* also brought to light the need to reduce the design time and implementation time of FPGA, possibly by parallelizing the process to use heterogeneous clusters to optimize its executable. Furthermore, it showed that design exploration tools were essential to automate performance estimation and code generation for multiple accelerators.

In 2010, *Novo-G* was presented as an experimental research cluster [110] consisting of 68 compute nodes built with COTS components. Its purpose was to help understand and advance the performance, productivity, and sustainability of future HPC systems and applications, focusing on the sustainability problem of current HPC systems using three different PCIe Intel FPGA boards: 24 nodes with 192 Stratix III FPGAs boards, 12 nodes with 192 Stratix IV FPGA boards, and 32 nodes with 128 Stratix V.

Novo-G has been used for several acceleration projects, ranging from biology to finance. One aspect that all applications have in common was being embarrassingly parallel and, therefore, naturally scalable. All of these applications were developed using the software offered as part of the *Novo-G* platform, and the results showed a huge speed increase compared to CPU clusters.

Chimera was the first work to focus on implementing an algorithmic FPGA and GPU pipeline. The *Chimera* cluster [16] was built using commercial components to explore alternative solutions to the computational constraints found in astronomy and provide access to HPC hardware for inexperienced users. The system is formed by CUs equipped with one CPU for management tasks, which is interfaced with 3 NVIDIA Tesla GPUs and 3 Altera Stratix IV FPGAs through PCIe via a backplane.

The success of *Novo-G* and the advancement of technology have allowed *Novo-G* to be upgraded to *Novo-G#* [111]. The cluster is made up of Gidel ProceV accelerators that house Stratix V FPGAs, two 8 GB DDDR3, and 32 Mbits of SRAM memory. The boards were interconnected by grouping 24 transceivers into six groups to support a torus topology with a total bandwidth of 300 Gb/s. The physical connection is done with fiber optics using QSFP+ modules. Data are transmitted via packets through a configurable single-level router network. This allows one to instantiate as many routers as necessary to service the ports and increase the internal bandwidth at the expense of the hardware resources. The network flexibility enables users to experiment with a variety of routing modalities, depending on the requirements of the application. *Novo-G#* nodes support three communication blocks: Low Latency, Custom, and Interlaken [112] to allow optimization of the physical layer depending on the application.

A common problem in custom computing is the lack of software development tools to help users build applications. To solve this problem, the *Novo-G#* team developed a modified Altera OpenCL to provide extended support for the 3D torus network present in the cluster.

An important aspect that most clusters left out, in addition to those focused on communication, was the interface with the physical world. This is the space that the *Axiom* platform [113] seeks to fill with a custom scalable cluster based on a board with a Xilinx MPSoC (Multiprocessor System on Chip) supporting the Arduino interface.

The MPSoC has an FPGA fabric, four 64-bit ARM cores for general-purpose applications, and two 32-bit ARM cores for real-time applications on the same die. Four USB-C ports managed by the FPGA MGTs are available for interconnecting the boards. A custom network interface (NI) in the FPGA provides support for all communications, allowing users to focus on their applications written on an OpenMP extension called OmpSs. The NI is divided into six main groups: a data mover that deals with DMA transfers, RX and TX controllers, and FIFOs to cache packets. A router is interfaced with each NI and is responsible for handling the USB-C channels, monitoring the network, and establishing virtual circuits.

As part of the *Axiom* project, a custom software stack [114] consisting of multiple layers was also developed. Its foundation is a distributed shared-memory (DSM) architecture. The main advantage of this approach is that it allows applications to directly address physical memory by transparently relying on an OS network. Several tests [115] and benchmarks have validated the effectiveness of the platform, pushing the project into IoT and edge computing [116].

Progress in this field has led to the creation of the Xilinx Adaptive Compute Clusters (XACC) [117] group under the Xilinx Heterogeneous Accelerated Compute Clusters (HACC) [118] initiative. This industry and academic collaboration focuses on the development of new architectures, tools, and applications for next-generation computers.

As part of this initiative, several clusters were built at some of the world's most prestigious universities in Switzerland, the US, Germany, and Singapore. At Paderborn University's National High-Performance Computing Center (PC2), high-performance clusters *Noctua* [119] and *Noctua 2* [120] were built to provide hardware to accelerate research on computing systems with high energy efficiency.

The *Noctua 2* cluster was designed to fit common server racks and be compatible with the standards of the network industry. It has 36 nodes with 2 AMD Milan. A combination of 48 Xilinx Alveo and 32 Intel Stratix 10 GX FPGAs comprised the reconfigurable computing part of the cluster. Each Stratix node has 4 plugable QSFP+ at 40 Gb/s and each Alveo has 2 QSPF+ at 100 Gb/s links and depends on Intel tools, such as *oneAPI* [121], *OpenCL*, and *DSP Builder*. A specific optical switch is used to build a configurable point-to-point network between all FPGAs.

More recently, *Enzian* [122] was developed as a scalable platform to fill the void left by industry-specific hybrid platforms. The reason behind *Enzian* is to provide a general, open, and affordable platform for research on hybrid CPU-FPGA computing, escaping the niche of specific-purpose hybrid platforms by providing a lot of flexibility. Explicit access to coherence messages, thermal and power monitoring, and an open baseboard management controller (BMC) allows for research that is not possible in any current commercial systems.

Similarly, *UNILOGIC* [123] presented a new approach, this time from the management of the cluster by introducing a Partitioned Global Address Spaces (PGAS) parallel model to heterogeneous computing. This allows hardware accelerators to directly access any memory location in the system, and locality makes coherency techniques unnecessary, greatly simplifying communication. By integrating Dynamic Partial Reconfiguration (DPR) into the framework, accelerators can be installed on the go. The *UNILOGIC* architecture was evaluated on a custom prototype consisting of 8 interconnected daughter boards, each with 4 Xilinx Zynq Ultrascale+ MPSoCs and 64 Gigabytes of DDR4 memory, yielding better energy and computing efficiency than conventional GPU or CPU parallel platforms.

In 2022, the supercomputer *Cygnus* [124] was updated [125] to follow a multi-hybrid accelerators approach based on GPUs and FPGAs. 32 Albireo nodes were added to *Cygnus*, each consisting of 4 NVIDIA V100 GPUs and 2 Intel Stratix 10 FPGAs. Similar to previous systems, a dedicated FPGA network was created with a 2D torus topology with improved stream capabilities, called CIRCUS [126]. Collaboration between the FPGAs and GPUs is achieved by using a DMA engine in the FPGA that accesses the GPU directly, bypassing the CPU, and offering almost double the throughput.

Finally, *Fugaku* [14], the first supercomputer to win all four categories in the Top500, presented a prototype FPGA cluster, *ESSPER* [127]. Motivated by the impressive continuous improvements in FPGAs regarding energy and performance, a cluster of 8 nodes, each with 2 Intel Stratix 10 FPGAs, was built and tested. This cluster was interfaced with *Fugaku* using a novel approach called loosely-coupled, where a host-FPGA bridging network provides interoperability and flexibility to all nodes in *Fugaku*.

Table 1.5 resumes the most relevant general-purpose clusters and shows their contributions along with the impact on power consumption and performance.

Table 1.5: General-purpose clusters' contributions, reported power and performance gains

Work	**Contribution**	**Power**	**Performance**
Maxwell [103]	One of the first general-purpose FPGA supercomputers.		10x to 100x faster than Intel Xeon CPUs.
Spirit [96]	Prototype FPGA cluster to explore methodologies for future PetaFLOP computers.		

Continued on next page

Table 1.5: *Continued from previous page*

Work	**Contribution**	**Power**	**Performance**
BEE3 [128]	Industry and academia's reconfigurable supercomputing ecosystems.		More than 10x faster when simulating systems with 16 processors compared to a software implementation.
Cube [13]	Low-cost architecture for scalable FPGA clusters with a single-configuration multiple-data programming paradigm for systolic arrays.	600x more efficient than a 180 dual Xeon quad-core cluster.	8x faster than a 359 high-end CPU cluster.
QP [105]	A heterogeneous cluster to explore multi-accelerator collaboration within a new framework.		FPGA part speedup of 48.4x with respect to a 16 AMD dual-core cluster.
Axel [15]	MapReduce framework for asymmetric parallel computing on heterogeneous clusters.	19x increase in efficiency was obtained for the FPGA and GPU implementation when compared to CPU.	The FPGA and GPU collaborative approach resulted in a 44.5x speedup for GARCH asset simulation compared to CPU-only implementation.

Continued on next page

Table 1.5: *Continued from previous page*

Work	**Contribution**	**Power**	**Performance**
Novo [110]	Experimental cluster to investigate productivity, performance, and sustainable architectures.	8 kW	More than 800x faster on bioinformatics applications over an AMD Opteron CPU.
Chimera [16]	Evaluation of a scalable heterogeneous desktop platform in the context of Berkeley dwarfs.		Peak performance 4.3 Gflop/J for the GPU part and 25 Mflop/J for the FPGA part.
Novo G#[111]	Reconfigurable cluster with reprogrammable interconnects to explore ways to minimize communication latency between nodes.		For 3D FFT, the cluster matches the latency of 512-core BlueGene/Q.
Axiom [113]	Scalable reconfigurable physical computing for low-cost applications.	8x more efficient for iris plus voice recognition than software implementation.	1.5x faster than a pure software implementation.
Noctua [119]	Data-center-grade FPGA cluster with optically switched interconnects for dynamic topology configurations.		Peak performance of 8.2 PFLOPS.

Continued on next page

Table 1.5: *Continued from previous page*

Work	**Contribution**	**Power**	**Performance**
Enzian [122]	Open hardware platform for CPU / FPGA systems with coherent access to asymmetric cache and fine-grained profiling.		
Unilogic [123]	First implementation of a Partitioned Global Address Spaces parallel computing model for heterogeneous clusters.	At least 10 times more efficient than CPUs or GPUs.	More than 2.5x faster than CPUs or GPUs.
Cygnus [125]	First hybrid cluster with direct collaboration of FPGA-GPU over DMA.		Radiative transfer simulation results show a speed-up of 12.8x for the FPGA-GPU collaborative approach over the GPU-only solution. Peak performance 2.4 PFLOPS.
ESSPER [127]	Loosely-coupled FPGA cluster prototype upgrade of Fugaku supercomputer in the world.		

1.1.5 Communication systems infrastructure

Another field of application where clusters of FPGAs are relevant is the emulation of the communication system infrastructure. The most important difference from many-core emulation is the need to interface with analog systems. This requirement implies providing additional external ports to interface with radio front-ends.

One of the first implementations was the *Berkeley Emulation Engine (BEE)* [129] in 2003. Its main purpose was to support design space exploration for real-time algorithms, focusing mainly on dataflow architectures for digital signal processing.

BEE was designed to emulate the digital part of telecommunication systems and to provide a flexible interface for radio front-ends. Computations are performed inside *BEE* Processing Units (BPU). Each BPU has a main processing board (MPB) and 8 riser I/O cards for 2400 external signals. The MPBs are the main computing boards that host 20 Xilinx Virtex FPGAs, 16 zero-bus turnaround (ZBT) SRAMS, and 8 high-speed connectors. FPGAs on the periphery of the board have off-board connectors to link other MPBs. A hybrid network consisting of a combination of a mesh network and partial crossbar, called a hybrid-complete graph and a partial crossbar (HCGP) [130], was implemented. A single-board computer (SBC) running Apache web services over Linux allows users to deploy their applications and perform configuration and slow control tasks.

To take full advantage of the platform, an automated high-level workflow was used [131] that relied on *MATLAB* and *Simulink* to develop the main hardware blocks. The BEE compiler then processes the output and generates the required VHDL files for the simulation and configuration of the system. A time-division multiple access (TDMA) receiver was fully implemented to satisfy real-time requirements and validate the workflow.

Following the *BEE* success, the *BEE2* [132] was conceived as a universal, standard reconfigurable computing system consisting of 5 Virtex 2 FPGAs, each with 4 DIMM connectors for up to 4GB of RAM. Four FPGAs are available for computing, and one was reserved for control tasks. Pivoting away from the HCGP, an onboard mesh was implemented between the 4 computing FPGAs. Using high-speed links, it was possible to aggregate the 5 FPGAs and use them as a single, larger FPGA. The workflow remained almost the same for *BEE2*, with the main change being the use of a computational model of synchronous data flow for both the microprocessor and FPGA.

To overcome the shortcomings of *BEE2* and take advantage of the already validated Spirit architecture [133], a digital wireless channel emulator (DWCE)[134, 135] was developed. It consisted of 64 nodes in the same way as Spirit, but with valuable upgrades to demonstrate the capabilities of FPGA clusters with military radios. Its capabilities improved with an upgraded FPGA, additional 2 FMC connectors, and the adoption of a standard MicroTCA.4 form factor.

Taking into account the possible improvements to BEE and because it was being developed as part of the research accelerator for multiple processors (RAMP) community [136], a fast response was presented in the form of *BEE3* [128]. The development of *BEE3* differed from previous iterations and successfully demonstrated a new collaboration methodology between industry and academia [137].

The architecture of *BEE3* changed substantially from that of its predecessor by removing the control FPGA and introducing a control module on a smaller PCB. Another important aspect worth highlighting is that, for the first time, a PCB was intentionally developed to support different FPGA parts, all interconnected using a DDR2 interface in a ring topology.

The *BEE3* prototype had approximately 30 collaborators, most of whom were professionals with extensive knowledge of computer-aided design (CAD). Relying on industry specialists for PCB design has resulted in simpler and more reliable PCBs within a shorter project time horizon. Additionally, it was possible to parallelize the design process, allowing the academic community to focus on firmware development.

The BEE collaboration presented its final iteration in 2010, consisting of *BEE4* and *miniBEE* [138, 139]. *BEE4* was updated to support Virtex 6 FPGAs and up to 128 GB of DDR3 RAM per module. The QSHs connectors were replaced with FMC connectors to support a wider range of mezzanine boards. *BEE4* was built around the Honeycomb architecture using the Sting I/O intermodule communication protocol. The design tools were further refined to include Nectar OS and BEECube Platform Studio in MATLAB/Simulink, which are unfortunately proprietary. However, being a proprietary system did not discourage its use in academia [140]. The success of *BEECube* attracted further interest from the industry, and was bought by National Instruments in 2015 [141]. Today, it is part of the FlexRIO [142] line-up, and software development is supported by proprietary tools. From this point onward, almost all implementations depend on commercially available emulation platforms.

To demonstrate the scalability of such implementations, the world's largest wireless network emulator was built, *Colosseum* [143], which can compute workloads of 820 Gb/s and perform 210 T operations per second. It was formed by CUs that consisted of three FPGAs in a chain. The outer FPGAs were used to interface with the radios and provide some processing. The central FPGA is dedicated to digital signal processing. Commercially available solutions were selected to avoid complications when designing the custom board. For radio-attached FPGAs, 128 USRP-X312 [144] software-defined radios were used. Each provides the analog interfaces required for the antennas, along with a Kintex 7 FPGA. As dedicated processing FPGAs, 16 ATCA-3671 [145] modules were used, each hosting 4 FPGAs. The 64 processing FPGAs were interconnected in a $4 \times 4 \times 4$ HyperX topology [146], allowing the data to be efficiently distributed for processing.

The ATCA-3671 modules are based on the BEE architecture and support the same development tools. Given the complexity of the system, a Python dataflow emulator [147] was built to confirm the topology and architecture of the system. It is possible to confirm the latency of the system by providing models of the implemented components and topology.

Another notable contribution of this study is the proposal of a data flow methodology [148]. It comprises three guiding principles that highlight the issues present in other implementations. The first principle is the use of a unified interface for modular components to favor portability. Second, when dealing with heterogeneous systems, the suggested approach is asynchronous processing to decouple operations from time and favor parallelization. Finally, based on design best practices, solutions are urged to be vendor-independent.

A summary of the most relevant implementations in the domain of communication systems infrastructure is presented in Table 1.6 with their contributions, power consumption metrics, and performance gains.

Table 1.6: Communication systems emulation clusters' contributions, reported power and performance gains.

Work	**Contribution**	**Power**	**Performance**
BEE [129]	Custom multi-chip FPGA emulation engine for ultra-wideband and multi-channel multi-antenna radio research.		More than 100k times faster than the best simulation software at that time.
BEE2 [132]	High-level block diagram design environment based on MathWorks Simulink and Xilinx System Generator library for one unified computation model for microprocessors and FPGAs.	100x more efficient than Intel Pentium 4 CPUs.	10x more computing throughput than a DSP-based system.
DWCE [134]	FPGA cluster for decentralized signal processing for radio emulation scalability.		Same performance as 15k CPUs cores running at 2 GHz.
Colosseum [143]	A scalable architecture to implement real-time channel emulators using software-defined radios and FPGA with a hardware-in-the-loop approach.		Emulation of more than 65k wireless channels with an area of up to 1 km^2.

1.2 Summary

In this chapter, relevant implementations of FPGA-based clusters are presented. Each cluster was described in detail with a focus on the contributions that were made. On top of this, the most impressive results from each platform are recounted to emphasize the potential of heterogeneous computing. The contributions go further than showing the potential of the different approaches, but also provide useful classifications. From these it can be appreciated that improvements in performance and energy efficiency are real and of orders of magnitude when compared to clusters based on CPU or GPU. However, this does not suggest that FPGAs alone are the answer to high-performance computing scaling, but rather the importance of the correct utilization of all technological resources while taking advantage of several levels of optimization provided by each platform. This is a complex challenge that requires a programming paradigm that efficiently and seamlessly integrates various CEs to allow users the maximum possible throughput by exploiting the physical characteristics of each CE.

Chapter 2

Reconfigurable cluster design

The state-of-the-art provides an ample view of the evolution of the heterogeneous supercomputing field. With this broad perception, it is hard to focus on the important design decisions for each cluster. One way to make it easier to drill into the details of the implementations is to divide the clusters into three domains, namely hardware, software, and network. These three domains with well-defined boundaries provide a scaffolding for designers, allowing one to weight different trade-offs related to design decisions. The relationship that each domain has with the others follows a layer model in which each layer provides services to the others. By correctly defining services and data flow, designers are better prepared for critical decisions when optimizing their clusters in the planning and design phases.

The proposed segmentation of the cluster infrastructure into three domains is shown in Figure 2.1. The first aspect, the network, covers the physical interfaces, communication protocols, and topologies chosen to interconnect the nodes. Another important aspect to consider is the hardware available in the CU. Each CU can have more than one CE type. Finally, software tools that allow the cluster to be securely available to users for development have to be considered. They encompass isolation tools that protect hardware from misbehavior, which are discussed in [149, 150] and go beyond programming languages, APIs, libraries, etc. Depending on the target application, these tools vary in scope, flexibility, and complexity. With a study of all previous contributions, it is possible to build a broad base that helps to understand the greatest challenges, future trends, and real capabilities of heterogeneous supercomputing, and also to make the correct design decisions understanding the compromises to maximize flexibil-

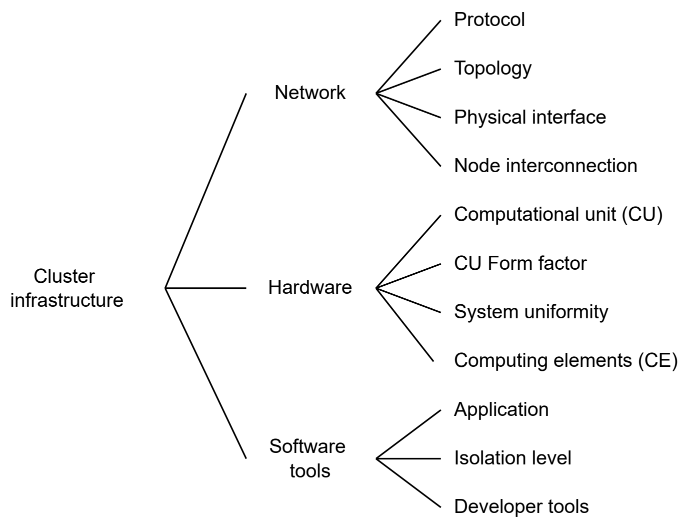


Figure 2.1: Main domains for the proposed segmentation of the cluster infrastructure.

ity. All of this is fundamental for making the correct design decisions during the planning and construction of the HyperFPGA.

2.1 Network

A cluster is no more than a set of computational elements (CE) that collaborate toward a common goal. The collaboration method and its means are crucial to ensure implementation efficiency. The means of collaboration extend from the hardware interfaces to the communication protocols and, ultimately, to the schedulers or other methods of synchronization. With this consideration, we can draw a line between systems that delegate communication tasks to an external entity (indirect) and those that incorporate the network stack inside the CE (direct). Another important aspect of the physical link is how it is used, in other words, the communication protocol. For example, in high-speed stream computing, it is desired that communication will be established as direct data channels with back pressure, which is particularly difficult to replicate with purely routed networks that rely on packets.

The physical link is discriminated according to the existence of any additional hardware that processes, redirects, or interprets a stream of data or packages between adjacent nodes as an indirect interconnection. This implies that a direct link is such that one node can interact directly with the nearest neighbor without the need for additional networking hardware. In a direct interconnection network, services are provided by in-fabric routers and switches, allowing users to experiment with different protocols at the expense of resources. This is particularly crucial in implementations that target heavy communication problems that require low latency. However, the dependence on external hardware also offers advantages. As in the case of *ARUZ* [41], which effectively interconnects thousands of CEs showing that indirect physical links are capable of extending scaling capabilities.

Furthermore, [151] provides an important comparison between the two physical link approaches. To compare the impact of both networks, a leaf-spine topology was implemented for the indirect switched network and a ring topology was implemented for the direct physical link. The switched network was modeled for 2048 FPGAs using 64 radix switches. These simulations showed that for a small message size ($\approx<$ 1 MB), a direct network offers a shorter transmission time than a switched network, regardless of the number of nodes. In contrast, larger payloads ($>$ 227 MB) benefit from a switched network, but only up to 1024 nodes when the direct network transmission time catches up. These results show that one approach is not necessarily better but that it comes down to the specific purpose of each cluster.

In addition, an architectural classification was proposed in [14]. This classification describes 4 different ways of integrating FPGA with other processing elements. All 4 different methods are based on the accessibility of the resources and the closeness between FPGAs and other processing elements.

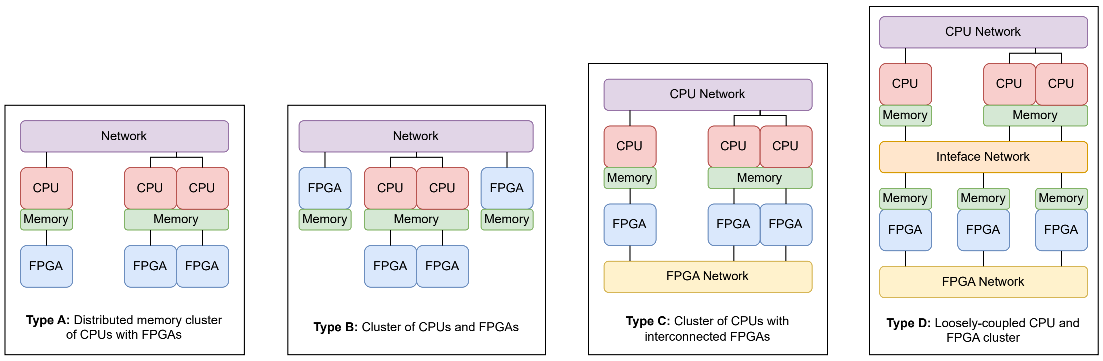


Figure 2.2: *ESSPER* [14] cluster classification based on the way the CPUs and FPGAs interconnect in a given implementation.

Figure 2.2 shows the proposed classification that consists of four different cases. Type A networks describe a cluster in which the FPGAs are directly attached to the CPUs. This is the most simple case for SoC-FPGAs or PCIe-mounted FPGAs. It has the limitation of constraining communication to a single network where all communications compete for the same bandwidth. As long as a data partitioning model that requires minimum communication between nodes is implemented, the penalties of the limited throughput might not be noticeable. The type B case shows one possible implementation of disaggregated FPGAs. These have the advantages of having FPGAs at the same level as CPUs and a uniform network infrastructure. Given the unbalanced throughput and latency of the devices, the network can be subjected to unexpected conditions that could decrease the final performance. This design can be found in cloud implementations such as *IBM CloudFPGA* [86] and *Catapult* [38]. The two remaining cases propose an FPGA dedicated network which allows for direct FPGA-to-FPGA communication without the interference of CPU communication. For type C, the implementation of the first case is extended to the dedicated FPGA network. Although it is a great improvement given that the network can be tuned specifically for the FPGAs, it has the disadvantage of grouping the CPUs to their attached FPGAs nonetheless, at least one CPU has to be present in the FPGA for control and configuration tasks. Finally, the most flexible approach is the type D network. It consists of a multiple network system in which the CPUs and FPGAs have dedicated networks, and an interface network which provides interconnectivity between the two systems. However, flexibility is paid for with the overhead in the cost and maintenance of a third network. This model is present in *ESSPER* [14], which is planned to provide FPGA resources to the existing Fugaku supercomputer.

Similarly, the topology of the clusters is a decisive design factor. The network topology describes the way nodes are interconnected. It can be physical when the topology defines the placement of the components or logical when it illustrates how the data flows in the network. Given that it is desired that nodes have the highest throughput with the lowest latency, most researchers have chosen a tightly interconnected topology, such as a mesh or a two-level mesh (TLM) [48, 129, 132, 72, 16, 113, 152, 79, 80]. These are great for providing consistent latency between nodes in the system, but they strongly impact scalability.

Other popular topologies are the 2D torus [65, 103, 96, 38, 76] and the 3D torus [53, 134, 111, 41]. They have the advantage of limiting the longest distance between nodes; however, as mentioned before, this distance continues to grow as more nodes are added to the system. There is also a restriction on how the system scales, given that it must keep a uniform shape and nodes should be added to fill columns or rows to avoid introducing inconsistencies in latency. Topologies such as BFT, systolic array, and star are application specific, since they restrict the connectivity of the nodes and most of the time require specific hardware, such as in the case of *BiCoSS* [80], *Cube* [13] and *COPACOBANA* [12].

Later works, such as *Noctua* and *Enzian* [122], are not bound by a fixed topology. In particular, *Noctua*'s [119] infrastructure provides an optical switch capable of implementing different topologies in runtime based on user requests. Naturally, since this is an external device, it can be implemented in any indirectly connected cluster. For some of the directly interconnected clusters, it may be possible to add an external switch, but only if a standard protocol and interface are used, such as *Novo-G* [110] and *Novo-G#*[111].

Standard protocols tend to place a strong dependence on physical interfaces. For this reason, most MGT implementations are based on the Aurora protocol or similar for the physical layer, and the data link relies on Ethernet. However, *Bluehive* [72] challenges this reasoning by implementing eSATA over PCIe connectors because PCIe was the only high-speed connector available on the FPGA nodes. On top of this, more drastic approaches implement a custom communication stack from the ground up, such is the case of *BEE* [129], *BEE2* [132], and *BEE3* [128] that use custom DDR protocols over GPIOs on same-board communications. Similarly, MGT implementations can benefit from custom protocols as demonstrated by *Novo G* [110] and [1] where optimizations to favor latency were performed at the physical level. Although there are benefits to be obtained by developing a specific software stack, it can be appreciated that, considering the complexity of bringing up one of these systems, standard protocols and interfaces have been more favored. This stems from the fact that proven technologies shorten design times, allowing developers to focus on other issues.

Table 2.1 shows in detail the characteristics of the clusters studied for each of the aspects discussed previously. The platforms are listed in ascending order by date.

Table 2.1: Network infrastructure.

Work	**Link**	**Topology**	**Interface**	**Protocol**	**Type**
RAW [48]	Direct	TLM	GPIO	Custom	B
BEE [129]	Indirect	TLM	GPIO	Custom	B
BEE2 [132]	Direct	Mesh	MGT	Internal DDR, External Infiniband	B
COPACOBANA [12]	Direct	Star	GPIO	Custom	C
FAST [10]	Direct	Star	GPIO	Custom	B
Janus [65]	Indirect	2D Torus	GPIO	Internal Custom, External GbE	B
Maxwell [103]	Direct	2D Torus	MGT	Custom	B
Spirit [96]	Direct	3D Torus	MGT	Aurora	C
BEE3 [128]	Direct	Ring	GPIO, MGT	Internal DDR, External GbE	B
Cube [13]	Direct	Systolic	GPIO	Custom	B
QP [105]	Indirect	Star	MGT	Ethernet, Infiniband	A
Axel [15]	Indirect	Star	MGT	Ethernet, Infiniband	C

Continued on next page

Table 2.1: *Continued from previous page*

Work	Link	Topology	Interface	Protocol	Type
BEE4 [139]	Direct	Ring	GPIO, MGT	Internal DDR, External GbE	B
Formic [55]	Direct	3D Torus	MGT	Sata	C
Novo G [153]	Direct	Systolic	MGT	Custom	C
Raptor [11]	Indirect	Star	MGT	PCIe	B
Tse et al. [70]	Indirect	Star	MGT	Ethernet	A
DWCE [135]	Direct	3D Torus	MGT	Aurora	B
Bluehive [72]	Direct	Mesh	MGT	PCIe	B
Chimera [16]	Indirect	Mesh	MGT	PCIe	A
Cuteforce [69]	Indirect	Star	MGT	Infiniband	A
Catapult [38]	Direct	2D Torus	MGT	Serial Lite III	C
Janus 2 [76]	Indirect	2D Torus	MGT	Internal Custom, External GbE	B
Superdragon [71]	Indirect	Star	MGT	Infiniband	C
Novo G# [111]	Direct	3D Torus	MGT	Custom	C
Axiom [113]	Direct	Mesh	MGT	Custom	C
IBM cloudFPGA [88]	Indirect	Star	MGT	GbE	B
ARUZ [41]	Indirect	3D Torus	GPIO, MGT	Internal Custom, External GbE	A

Continued on next page

Table 2.1: *Continued from previous page*

Work	**Link**	**Topology**	**Interface**	**Protocol**	**Type**
Greenflash [152]	Indirect	TLM	MGT	Internal PCIe, External GbE	C
Noctua [119]	Direct	Flexible	MGT	Custom	C
Astrobyte [79]	Direct	Mesh	MGT	Custom	B
Enzian [122]	Direct	Flexible	MGT	Internal PCIe, External GbE, Custom	C
UNILOGIC [123]	Direct	Hypercube	MGT	AXI Chip-2-Chip	C
BiCoSS [80]	Direct	BFT	GPIO	Custom	C
Coloseum [143]	Direct	Ring, Hypercube	GPIO, MGT	Internal DDR, External GbE	B
Cygnus [125]	Direct	2D Torus	MGT	CIRCUS	C
ESSPER [127]	Direct	2D Torus	MGT	Internal Avalon-ST, External custom	D

2.2 Hardware

The hardware of a cluster can be divided into computational units (CU), which are any entity that is available for computing and is the smallest independent functional part of the cluster. According to this definition, devices that act as pure network appliances, routers, and switches are not considered. A CU can be made up of multiple CEs (CPU, GPU, or FPGA). Over time, smaller CUs are preferred when dealing with general-purpose clusters, whereas specific problems can benefit from larger CUs with an ad hoc network topology. Another critical aspect of the CU is its form factor or physical shape. A custom form factor will require custom support structures and cooling solutions, while standard form factors can benefit from existing hardware.

In the context of the *Axel* [15] cluster, a classification structure was proposed. It focuses on assigning a class to CUs based on the heterogeneity of their CEs. Four different classes were considered, as shown in Figure 2.3. The Uniform-Node Uniform-System (UNUS) corresponds to a homogeneous cluster and is typically formed by CPUs. However, we can also find FPGA implementations such as Formic [53], Janus I [65], and Janus II [76]. Uniformity significantly simplifies the management, programming model, and maintenance of clusters. Interestingly, the advantages have not been critically outweighed by the disadvantages, since several other studies have explored other more complex approaches. Performance-wise, there is a lot to win when dealing with non-uniform nodes. As shown in Table 1.1, all computing problems can be classified into 13 categories. By carefully studying the affinity between each category of problems and the resources encapsulated in each CE, ideal candidates can be found to solve each problem. This means that, by mixing and matching, a heterogeneous cluster may offer performance advantages over a homogeneous cluster.

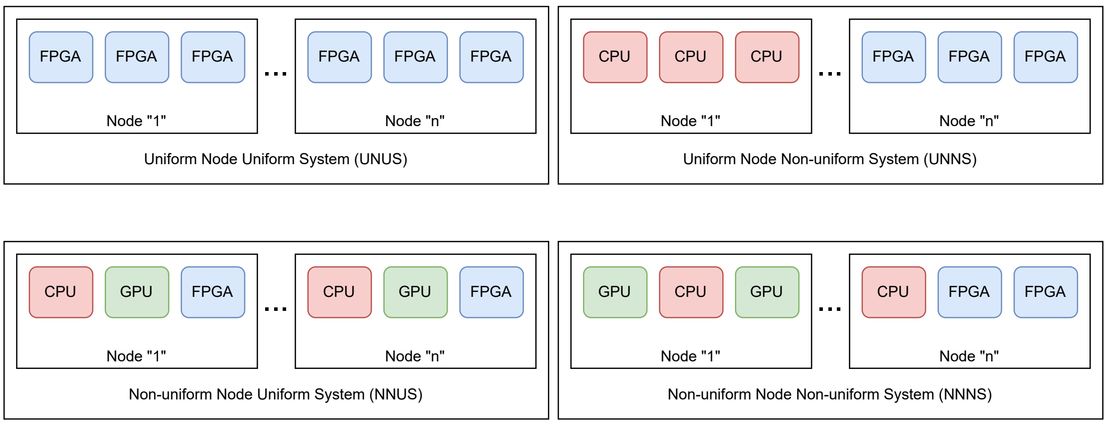


Figure 2.3: *Axel* [15] node or computational unit (CU) classification showing possible uniform and non-uniform node and system configurations for heterogeneous clusters of CPUs, GPUs, and FPGAs.

Figure 2.4 shows how each of the problems (dwarfs) is assigned to CEs (CPUs, GPUs, or FPGAs) depending on their characteristics [16]. This map shows that there are clear advantages in using one type of CE over another, mainly between GPU and FPGA. This is further supported by the implementations of *Chimera* [16] and *Green Flash* [152]. In both cases, GPUs and FPGAs are intended to be used as collaborative CEs. However, this new paradigm makes development much more complicated, not because of the lack of tools for FPGA and GPU co-processing, but also because of the required radical change of mindset when leaving the traditional CPU plus accelerator context. This is reflected in the reported applications of *QP* [105] in which users used only a combination of CPU plus GPU or FPGA.

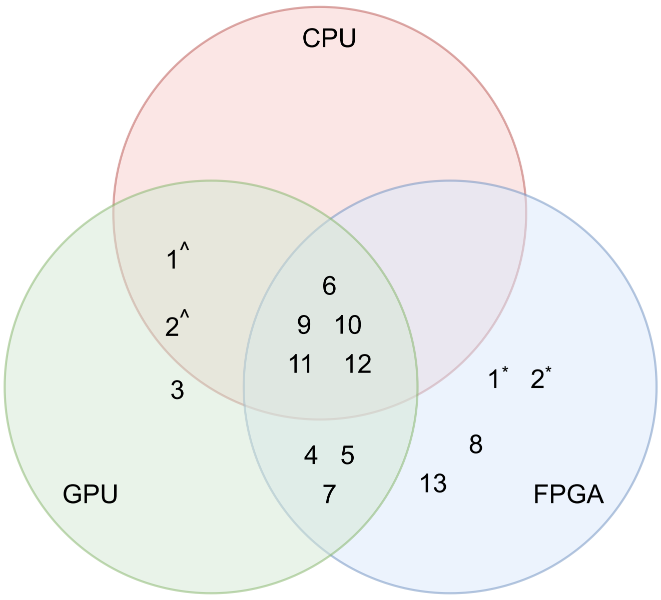


Figure 2.4: 13 dwarfs (Table 1.1) mapped-out according to highest affinity computational element (CE); the superscript ˆ refers to floating point and $*$ to fixed point [16].

As previously described, classification based on node (CU) and system uniformity is useful to understand the programming paradigm. According to the previous definition of nodes, multiple CEs can be hosted on a single node. The balance between a crowded node or a simple node lies in the diversity of CEs and network infrastructure. Diverse CEs in a single node allow for the highest resource availability per CU for developers, but it remains a challenge to interface all devices considering all the different ports. This leads to different form factors that directly impact the way the cluster scales and, more importantly, the availability of physical structures to hold the nodes in place and provide efficient cooling. Custom CU form factors usually host several CEs and can compromise not only scalability, but also fault tolerance. This is the case for *ARUZ* [41], where a single node hosts up to 11 FPGAs. In the unfortunate case where one or more CEs break down, the OS must be notified to circumvent these nodes or completely ignore them until they are fixed. In this regard, COTS clusters have a great advantage: the up-bring cost is mainly absorbed by the industry by providing tested and validated nodes for quick installation, which is the case for *Noctua 2* [120] and *Catapult* [38], among others. Some rare cases of industry and academia collaboration greatly benefit from COTS advantages with specific research-motivated modifications, such as in *Novo-G*[110] and *BEE3* [137].

Table 2.2 shows the most relevant clusters studied, classified according to the characteristics of their CU. The *Axel* classification system was used to identify the uniformity of the node (CU) and system. The total number of CUs is also presented to demonstrate the scale of the cluster. Some studies have presented the architecture of a single node as a building block for a future cluster. These are considered relevant for their contribution to the study of heterogeneous workloads. The form factor of each work is shown in the respective column. Given that these are heterogeneous systems, different types of CE may be present in CUs and sometimes even among CUs.

Table 2.2: Hardware architecture of computation units (CU) with respect to their computational elements (CE).

Work	**Class**	**CU**	**Form factor‡**	**CE/CU**	**Resources**
[48]	UNUS	5	Custom	64	320 Xilinx 4013 FPGAs
[129]	UNUS	4	Custom	5	20 Xilinx XC2000E FPGAs
[132]	UNUS	40	Custom	5	200 Xilinx Virtex-2 Pro FPGAs
[12]	UNUS	20	Custom	6	120 Xilinx Spartan-3 S1000 FPGAs
[10]	NNUS	2	Custom	8	16 Xilinx Virtex-5 FPGAs
[65]	UNUS	16	2U HR	16	256 Xilinx Virtex-4 LX200 FPGAs

Continued on next page. ‡FR and HR stand for Full or Half 19 inch rack, respectively.

Table 2.2: *Continued from previous page*

Work	**Class**	**CU**	**Form factor‡**	**CE/CU**	**Resources**
[103]	NNUS	32	PCI-X	3	32 Intel Xeon; 32 Nallatech HR101s with Xilinx Virtex-4 LX160; 32 Alpha Data ADM-XRC-4FXs with Xilinx Virtex-4 FX100 FPGAs
[96]	UNUS	64	Custom	1	64 Virtex-4 FX60 FPGAs
[128]	UNUS	64	2U FR	4	256 Xilinx Virtex-5 LX155T FPGAs
[13]	UNUS	8	Custom	64	512 Xilinx Spartan-3 S4000 FPGAs
[105]	NNUS	16	Desktop PC	6	16 AMD dual-Core CPUs; 64 NVIDIA Quadro FX 5600 GPUs; 16 Nallatech H101 with Xilinx Virtex-4 LX160 FPGAs
[15]	NNUS	16	4U FR	3	16 AMD quad-core CPUs; 16 NVIDIA Tesla C1060 GPUs; 16 Xilinx Virtex-5 LX330T FPGAs
[139]	UNUS	1	2U FR	4	4 Xilinx Virtex-6 475T FPGAs
[55]	UNUS	64	Custom	1	64 Xilinx Spartan-6 LX150T FPGAs
[110]	NNNS	68	4U FR	4, 8, 16	192 Intel Stratix III E260 FPGAs, 192 Intel Stratix IV E530 FPGAs, 128 Intel Stratix V GSD8 FPGAs
[11]	NNUS	16	Custom	4	64 Xilinx Virtex-5 FX100T FPGAs
[70]	NNUS	8	2U FR	4	16 AMD Phenom 9650 quad-core CPUs; 8 NVIDIA Tesla C1060 GPUs; 8 Xilinx Virtex 5 LX330T FPGAs
[135]	UNUS	64	MicroTCA1		64 Xilinx Virtex-6 475SX FPGAs

Continued on next page. ‡FR and HR stand for Full or Half 19 inch rack, respectively.

Table 2.2: *Continued from previous page*

Work	Class	CU	Form factor‡	CE/CU	Resources
[72]	UNUS	4	8U FR	16	64 DE4 boards with Altera Stratix IV 230 FPGAs
[16]	NNUS	1	Custom	7	1 Intel i7 hexa-core CPUs; 3 NVIDIA Tesla C2070 GPUS; 3 Intel Stratix IV 530 FPGAs
[69]	UNNS	15	2U FR	1	1 NVIDIA GeForce GTX 680 GPUs; 14 Xilinx ML605 FPGAs development boards
[38]	NNUS	1632	1U HR	3	3264 Intel 12-core CPUs; 1632 Intel Stratix V D5 FPGAs
[76]	UNUS	16	2U HR	16	256 Xilinx Virtex-7 VX485T FPGAs
[71]	NNUS	64	2U FR	2	64 NVIDIA Tesla C2050 GPUs; 64 Xilinx Virtex 5 LX330 FPGAs; 128 Intel Westmere hexa-core CPUs
[111]	NNUS	64	4U FR	3	128 Intel Xeon E5 CPUs; 64 Intel Stratix V FPGAs
[113]	NNNS	16	Custom	1	16 Xilinx Ultrascale+ ZCU9EG SoC-FPGAs
[88]	UNUS	16	2U FR	64	1024 Xilinx Kintex UltraSCALE KU060 FPGAs
[41]	UNUS	2901	Custom	11	23040 Xilinx Artix-7 A200 FPGAs, 2922 Xilinx Zynq Z015 SoC-FPGAs
[152]	NNUS	5	ATX mini	3	5 Xeon Phi 7210 CPUs; 5 Intel Arria 10 SoC-FPGAs; 5 Intel Arria 10 FPGAs
[119]	NNNS	36	2U FR	5	72 AMD 64-cores Milan CPUs; 48 Xilinx Alveo U280 FPGAs; 32 Stratix 10 FPGAs

Continued on next page. ‡FR and HR stand for Full or Half 19 inch rack, respectively.

Table 2.2: *Continued from previous page*

Work	**Class**	**CU**	**Form factor‡**	**CE/CU**	**Resources**
[79]	UNUS	4	Custom	1	4 Intel Stratix IV GX FPGAs
[122]	NNUS	1	2U FR	2	Cavium ThunderX CN88XX CPU; Xilinx Ultrascale+ VU9 FPGA
[123]	UNUS	16	1U FR	4	64 Xilinx Ultrascale+ ZU9EG SoC-FPGAs
[80]	UNUS	5	Custom	7	35 Intel Cyclone IV FPGAs
[143]	UNUS	16	2U FR	28	128 Intel 48-core Xeon CPUs; 128 NVIDIA Tesla K40m GPUs; 128 Ettus USRP X310 with Xilinx Kintex-7 410T FPGAs; 64 Xilinx Virtex-7 690T FPGAs
[125]	NNUS	32	4U FR	8	64 Intel Xeon; 128 NVIDIA Tesla V100; 64 Intel Stratix 10 GX2800 FPGAs
[127]	UNNS	8	1U FR	4	16 Intel Xeon quad-core CPUs; 16 Intel Stratix 10 SX2800 SoC-FPGAs

‡FR and HR stand for Full or Half 19 inch rack, respectively.

2.3 Software tools

Each work discussed would be incomplete if there were no tools available to help users develop their applications. These tools provide different layers of isolation, ranging from templates that encapsulate internode communication to complete operating systems that manage multiple user access. Each of the tools offers a degree of abstraction that includes all the underlying details to provide services to the user or to a higher layer. The depth of the tool stack depends on several factors:

- Purpose of the cluster

- Degree of freedom intended for the user (isolation)
- Cluster flexibility

A stack of tools can be structured according to the services provided and required, as shown in Figure 2.5. First, we have the interface with the external world at the physical level. Naturally, we rely on electrical and optical signals controlled by internal gates, GPIOs, or MGTs. Typically, in HPC, this is not out for discussion, given that CPUs and GPUs have fixed interfaces, but FPGAs are not bounded by this. Thus, the communication layer can be either available for users to freely customize and test or fixed by the developers and provided as a service. In addition, we have the CEs that can be CPU, GPU, or FPGA cores. It is in this part where the actual computing is performed, in the form of the ISA in the case of CPUs and GPUS or FBs in the case of FPGAs.

The OS provides drivers, file system, and a scheduler, among others, to interact safely with the hardware. This creates a safe space for users to build applications based on hardware and communication services. At this level, users must rely on a programming language that describes how the underlying parts cooperate for the intended computation. Some studies have presented new programming languages that aim to capture the different programming paradigms in heterogeneous clusters [92, 93, 154, 155, 156]. Tools that take the abstract description of the computation and transform it into instructions may be provided as a contained solution or along libraries and APIs to facilitate development. Another level of abstraction may be introduced, in which users interact with pre-built blocks in a fixed context inside a GUI.

Depending on the scope of the application, user needs vary and may require deeper access to the system or more abstract tools. Most general-purpose clusters are intended to be used as research platforms. This requirement relaxes many management applications and abstraction layers that, in turn, must be provided to the user in other cases. As research could participate in the lowest level of communication, users may need the freedom to change the electrical standard of the GPIOs or the encoding of the MGT. These properties are only available if the user sees the platform as a bare metal solution or if the development environment has a standardized way of defining communication devices. In any case, most systems avoid this by providing the user with a template that abstracts the communication layer. Specific application clusters seek to contain most of the details such that the user faces only the challenges related to the

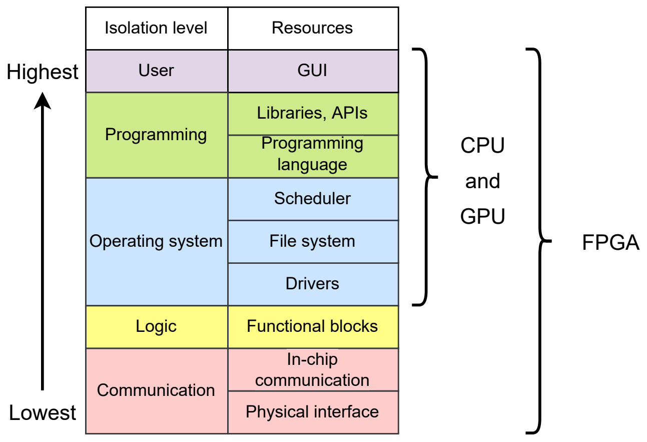


Figure 2.5: Tool stack divided into different levels depending on the user isolation from the hardware.

application.

The tool stack can also greatly benefit from being open [157]. In this regard, FPGA development frameworks have been significantly delayed as opposed to CPU and GPU. Currently, one can use complete open-source frameworks to develop applications for CPUs and GPUs, but FPGAs are radically different. One reason for this is that the stack of tools is fundamentally different. Instead of targeting fixed hardware through a well-known and well-defined ISA, FPGA tools target configuration memory with architecture-specific information that changes with vendor and family. These architectural details tend to be industry secrets that force developers to rely on proprietary tools with all their benefits and limitations. Efforts have been made to create completely open-source workflows such as Yosys [30] and F4PGA [31], in which experienced users can actively collaborate to improve the platform.

Lastly, an important aspect directly related to the application is the level of flexibility provided by the other network and hardware domains of the cluster. Some applications may be prepared to host external hardware to extend the capabilities of the CU, for example, analog to digital converters. This is the case for all *BEE* implementations [129, 132, 137, 139]. Other clusters may be capable of interchanging CEs, which would require the hardware to be portable, meaning that updates to the hardware would not force modifications of the development tools. Similarly, the network domain can provide flexible topologies [119] and communication protocols

[111].

Table 2.3 shows the development tools provided and the intended application of the studied clusters. This table shows each cluster with its intended application along with the software tools and the degree of isolation provided.

Table 2.3: Target application and development tools.

Work	**Application**	**Development tools**	**Open Source**	**Isolation level**
[48]	Manycore emulator	Custom tools		Programming
[129]	DSP emulator	BEE Compiler		Programming, logic
[132]	RF emulation	BEE Compiler		Programming, logic
[12]	Cryptography	Copacobana Lib		Programming
[10]	Manycore emulator	FAST Software toolbox	✓	User
[65]	Spin-system simulator	Joslib, Josd, JOS	✓	User
[103]	General	Parallel ToolKit		User
[96]	General			
[128]	General	BEE Compiler		Programming, logic
[13]	General	Template		Communication
[105]	General	OFED, Phoenix framework		User
[15]	General	Custom		Programming
[139]	General	BEE Compiler, Nectar OS, BEE platform studio		Programming, logic

Continued on next page

Table 2.3: *Continued from previous page*

Work	Application	Development tools	Open Source	Isolation level
[55]	Manycore emulator			Programming
[153]	General	OpenCL, Custom tools		Logic
[11]	General	RAPTORLIB, RAPTORAPI, RAPTORGUI		Programming
[70]	Monte-carlo simulations			User
[135]	Digital wireless emulator			
[72]	Neural network simulator	Bluespec		
[16]	General			Programming
[69]	Cryptography			User
[38]	Bing search engine			Logic
[76]	Spin-system simulator	Joslib, Josd, JOS	✓	User
[71]	Cryo-electron microscopy			User
[111]	General	LEAP OS, Custom tools, Bluespec	✓	Logic
[113]	General	OmpSs@FPGA	✓	Programming

Continued on next page

Table 2.3: *Continued from previous page*

Work	Application	Development tools	Open Source	Isolation level
[88]	FaaS	OmpSs@cloudFPGA	✓	Logic, Programming
[41]	General	Custom tools	✓	Logic
[152]	Astronomic imaging			User
[119]	General	Custom tools	✓	Communication
[79]	Neural network simulator			User
[122]	General	Coyote OS	✓	Logic
[123]	General	UNILOGIC, Hardware Accelerator Controller		Programming, logic, communication
[80]	Neural network simulator			User
[143]	Digital wireless emulator	BEE Compiler, RFNoC, UHD	✓	Programming, logic
[125]	General	CIRCUS library and API		Programming, logic
[127]	General	R-OPAE, DMA library, AFUShell		Programming, logic

2.4 HyperFPGA design

The HyperFPGA [158] has been conceived to be an experimental testbed for reconfigurable computing architectures, where one of the most crucial aspects is flexibility. Naturally, any general-purpose cluster should allow the user to express an algorithm in an efficient way that takes advantage of all available resources. Each of the domains (network, hardware, and software) presents opportunities for optimization which can only be profited from if the domain offers them as services to the user. This puts a lot of pressure on the software tools since, with user intervention at various levels, the corresponding protections must be placed to avoid fatal mistakes.

From the ground up, the hardware that offers the highest level of flexibility is the one with the highest heterogeneity, demonstrated by the 13 dwarf affinity in Figure 2.4. Different cluster implementations are present in the literature, from multi-CE PCBs such as in *FAST* [10] to a more generic approach of using a desktop PC motherboard and mounting GPUs and FPGAs in PCIe slots, as was the case of *Axel*[15]. SoC-FPGAs provide a great level of heterogeneity in a convenient package that reduces power consumption and offers a single-chip solution that includes CPUs, GPUs, and FPGAs. These ICs also offer a convenient way to offload control tasks to already existing CPUs, allowing for a more performant implementation by liberating the FPGA from less critical tasks. Furthermore, by placing these chips on a smaller dedicated board with memory and power sequencing, the complexity of the PCB is constrained into a system-on-module (SoM). By using commercially available connectors and an open pin-out, SoMs are transformed into modules that can be swapped over different carrier boards depending on the application. This modularity and heterogeneity is ideal for a general-purpose platform such as the HyperFPGA.

However, choosing SoC-FPGAs as CE has an impact on the network infrastructure. In SoC-FPGAs, the FPGA is physically interconnected to the CPUs forming a distributed shared memory cluster of CPUs with FPGAs, shown in the first diagram of Figure 2.2. To avoid the pitfall of scarce throughput related to a type A system, the platform must be extended by adding an FPGA-dedicated network. As shown in [113, 110, 111], a dedicated FPGA network allows one to increase the flexibility of the platform and the overall performance of the cluster [159, 160].

Speaking of the network domain, it also has its own challenges. The first thing to define is the nature of the physical link. Whether direct or indirect, it impacts the cost of the implementation by including additional hardware in the indirect approach. Fortunately, this decision has minor effects on performance, as shown in [151]. Another advantage of choosing a direct physical link is that there are no restrictions on the protocol in addition to the physical constraints of the connectors. To further extend the freedom, the HyperFPGA should allow the user to choose the protocol of their liking, which can be achieved by interconnecting the FPGAs through their GPIOs and MGTs.

To take advantage of this freedom, a software environment that allows for deep interaction is required. As proven by the analysis in the previous section and the Strategic Infrastructure for Reconfigurable Computing Applications (SIRCA) [161] report, this is not an easy task. First, the diversity of users' expertise implies that the framework must be modular to enable power users to substitute any part of the stack with their desired implementations while maintaining compatibility and at the same time friendly to allow application developers to implement their algorithms. Second, given that the implementations may vary greatly depending on the parallel paradigm, *MPI* provides the generic enough approach to encapsulate parallel algorithms without enforcing hard constraints. Third, given that the HyperFPGA is an experimental platform, the administration must be as simple as possible and preferable open source for continuous development. All these decisions give shape to the HyperFPGA.

Table 2.4 summarizes the decisions taken by domain to build the HyperFPGA cluster.

Table 2.4: HyperFPGA design description by domain.

Hardware	
Class	NNUS
CU	16 +
Form-factor	Custom
CE/CU	1
Resources	SoM with SoC-FPGA
Network	
Link	Direct
Topology	2D Mesh
Interface	GPIO, MGT
Protocol	Internal custom
Type	C

Software tools	
Application	General-purpose
Development tools	Modular
Open Source	✓
Isolation level	Communication, logic, OS

2.5 Summary

The detailed study of the decisions taken in the design and implementation of other clusters offers valuable information. First, recognizing the three main domains and their borders allows one to identify where specific services should be placed for optimal implementation. Second, being able to identify the related domain to a certain service allows maximizing the flexibility of such a service by confining the configuration parameters in a modular manner. From this analysis, it was possible to determine the best way to proceed with the design and construction of the cluster, which is described in the following chapters.

Chapter 3

HyperFPGA hardware description

Following the discussion on hardware made in Chapter 2, in this chapter the description of the hardware that builds up the first HyperFPGA prototype is described. The HyperFPGA follows an NNUS approach consisting of a heterogeneous node that is uniform throughout the cluster, diverse enough to provide room for experimental architectures while keeping the simplicity of a uniform management system.

There are many devices that offer a high level of heterogeneity, from single-board computers to accelerated processing units that host CPUs and GPUs in the same die. However, there is no better example of single-chip heterogeneity than SoC-FPGA. They have the great advantage of being reconfigurable for the most part while having several hard fixed IPs ranging from CPUs to artificial intelligence engines on the most recent devices, such as the AMD Versal [162].

The incomparable level of heterogeneity and compartmentalization of SoC-FPGAs suffers from a common issue in the IC world, pinout incompatibility between families or vendors. To completely circumvent this issue, a SoM was considered, specifically the layout provided by Trenz [163]. By building around an open and standard SoM, it is possible to develop a platform that is modular, vendor-independent, and future-proof.

This chapter describes the HyperFPGA carrier board that hosts a SoM that follows an open pinout. The board constitutes a node or CU that can be interconnected to build clusters. By exploiting the integration of memory and power sequence in the SoM, the carrier board is greatly simplified, with a focus on interconnection, safety, and power monitoring.

3.1 PCB overview

The HyperFPGA was originally envisioned [158] as a 3D array of nodes placed in a cube. Therefore, the physical topology maximized node interconnection while minimizing the distance between all nodes by wrapping the edges in a 3D torus. This approach is quite similar to the implementation in [55], but showed the complication of scalability due to the requirement for all sides to be scaled equally to keep distance and latency uniform throughout the cluster. For these reasons, the first HyperFPGA prototype was built with scaling in mind.

The HyperFPGA carrier board is a square of 15 cm per side. On each side there is a high-speed high-density Hirose FX18 connector with 140 pins. These connectors allow for 2D uniform tiling and provide physical interconnection between the SoC-FPGAs in each SoM. In this implementation, the layout was simplified from a 3D torus to a 2D mesh network topology, while maintaining the flexibility of transforming into a torus with the help of adapter cards and cables.

The carrier board was designed to provide power, interconnections, monitoring, and booting facilities for the SoM. Additionally, a peripheral component interconnect express (PCIe) x4 port is used to increase SoC-FPGA capabilities. To provide access to the SoM, JTAG, Gigabit Ethernet, and USB to UART are available on the board. To increase the static memory capacities of the board, a micro SD card port was included. Micro SD cards provide an exchangeable storage medium that can be up to 1 terabyte in size at a low cost, fitted for boot files and a complete OS with user files. The carrier board with all these components is shown in Figure 3.1.

3.1.1 SoM description

In our specific implementation, the SoMs have an AMD Ultrascale+ SoC-FPGA consisting of a quad core A53 ARM processor, dual core R5 ARM real-time processor, and Ultrascale+ FPGA. With the SoC-FPGA, various speeds and sizes of SDRAM and two flash memory chips are available. The SoM also provides a 10-channel programmable phased-locked loop (PLL) clock generator and various power converters that handle the power sequencing of the whole system. These power converters split the power domains giving granularity that helps to monitor the power consumption of different aspects of a given application. From the FPGA 2 high-density

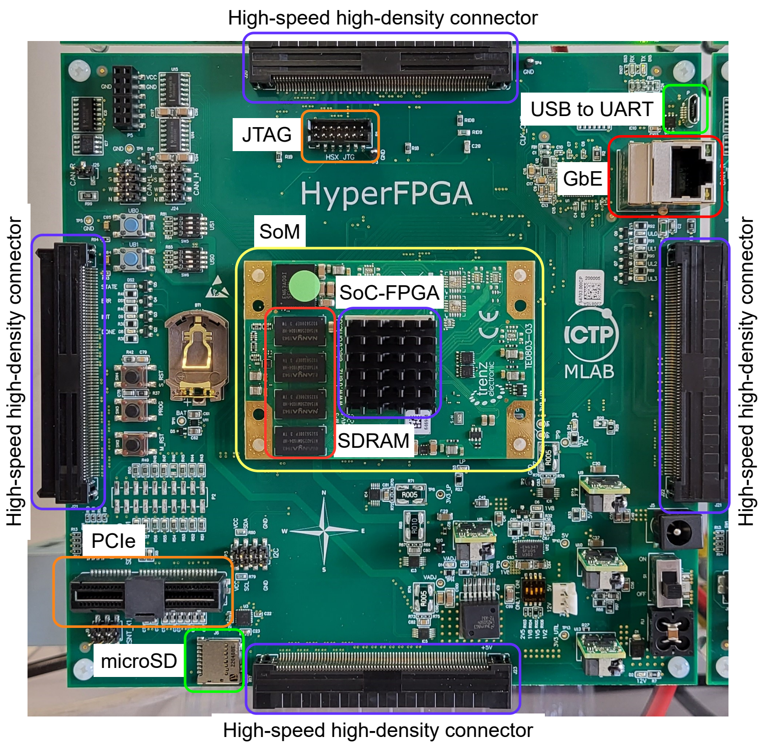

High-speed high-density connector
USB to UART
JTAG
GbE
HyperFPGA
SoM
SoC-FPGA
High-speed high-density connector
High-speed high-density connector
SDRAM
PCIe
microSD
High-speed high-density connector

Figure 3.1: HyperFPGA carrier with a System-on-Module, the most relevant parts of the system are highlighted with colored boxes.

(HD) I/O banks and 3 high-performance (HP) I/O banks are available along up to 16 MGT lanes. The CPU has a single MGT consisting of 4 lanes, which in the case of the HyperFPGA, are used for the PCIe x4 onboard connector. All described blocks are shown in Figure 3.2.

On top of this, four 160 pin connectors expose both peripherals and I/Os. Figure 3.3 shows the 4 connectors present in the SoM. Through these connectors, all the resources on the SoM are accessible from the carrier board.

Moreover, this pinout supports a wide variety of AMD MPSoC devices. Recently, Sundance [164] made available a SoM with a compatible pinout and form-factor hosting a Microchip Polarfire SoC [165]. These developments make the pinout a strong option for a standardized SoM interconnection. Furthermore, more connectors can be added by increasing the size of the SoM, extending the life of the platforms.

A wide range of AMD Ultrascale + SoC-FPGAs is supported in the same SoM format, the

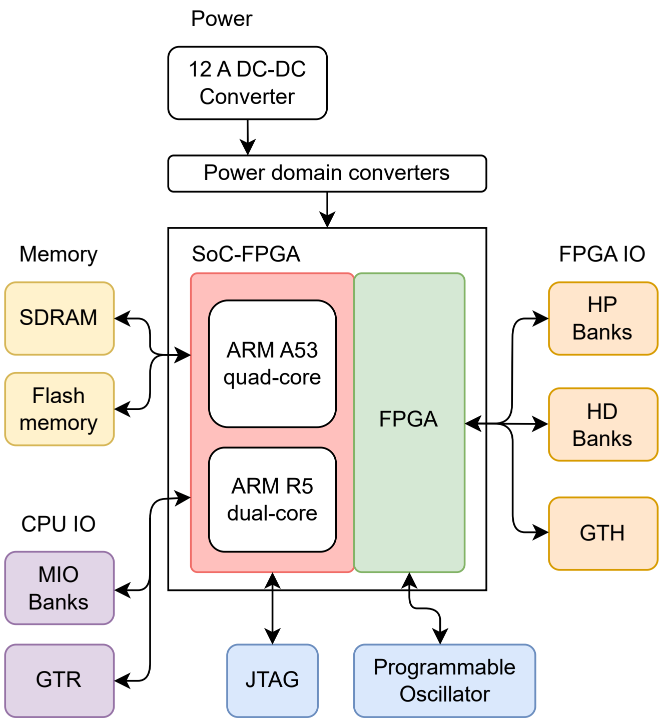


Figure 3.2: SoM block design, showing the SoC-FPGA, DDR memory, onboard oscillator, and power modules.

relevant differences being the external RAM, the amount of FPGA resources, and the gigabit transceiver (GTH) count, thus impacting the available communication bandwidth. Despite the differences, all AMD SoC-FPGAs are highly dependent on the proprietary Vivado [26] tool chain for bitstream generation.

3.1.2 Power and monitoring

Each carrier board has a barrel plug to be connected to a 12 V external power supply. The main power is filtered and limited through a resettable fuse at 15 A. The 12 V are further converted by 5 onboard direct-current to direct-current (DCDC) converters that supply each of the power domains present in the SoM. Additionally, the 12 volts provide power to a PCIe connector, which is exclusively available to the CPU for a Solid State Drive (SSD) or accelerators.

Each power domain is individually monitored using the INA226 [166] current and power

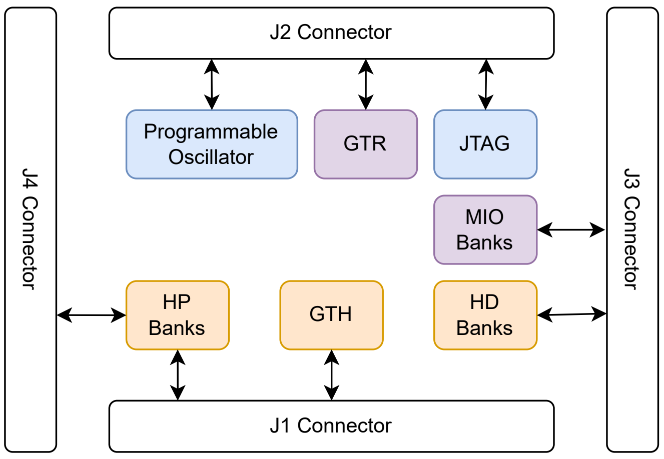


Figure 3.3: The 4 connectors of the SoM with the number of pins and details on the signals.

monitoring chip. In total, 4 INA226s are present on the board. All power monitors are read through a dedicated port using I2C. The monitored power domains are as follows:

- Low power peripherals and ARM R5 CPU
- Full power consisting of GTHs and ARM A53 CPU
- FPGA programmable logic
- Adjustable voltage for FPGA input/output buffers

The different power domains are shown in figure 3.4. As listed above, there are 4 independent power domains which serve different parts of the SoC-FPGA. By separating the power monitoring zones, we can obtain information on specific activities. One of the most crucial factors is the ability to measure the power consumption of off-board communication, particularly FPGA I/Os supplied by the adjustable voltage power domain. With this information, it is possible to adjust the implementation for better power consumption and performance. Furthermore, the ability to measure each computing domain separately, FPGA logic, R5 CPU, and A53 CPU, offers valuable insight into the real performance per watt of heterogeneous algorithm implementations.

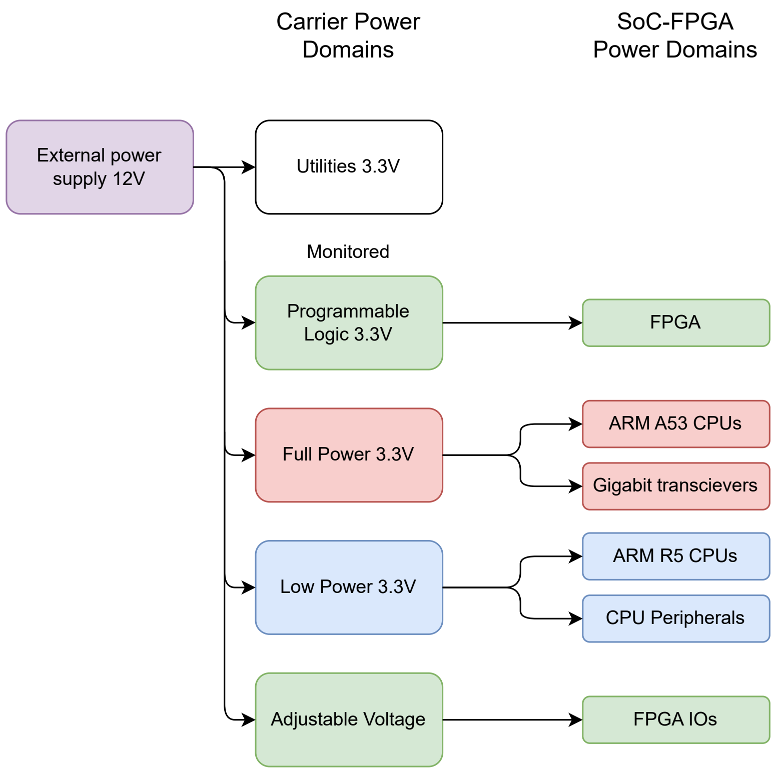


Figure 3.4: HyperFPGA power domains with their dependant components.

3.1.3 Carrier board interconnection

The carrier board uses four high-speed connectors to interface with four adjacent boards. The selected parts are the coplanar Hirose FX18 connectors. By placing four of these connectors, one on each side of the square, we provide 32 pairs of low-voltage differential signals with varying maximum speeds. The variety of possible electrical signals allows for the implementation of different protocols at the physical level.

Up to 4 GTH lanes are available per connector. Each GTH is capable of up to 16.3 Gbps in a point-to-point connection. Given that these rely on hard IP cores, vendors offer proprietary implementations that might make porting a design complicated. To alleviate this, solutions such as the Kyokko IP [167] provide an open source wrapper for transceivers, resulting in a portable and interoperable interface between both Intel and AMD FPGAs.

Figure 3.5 shows the three networks on the platform. ARM processors are connected through

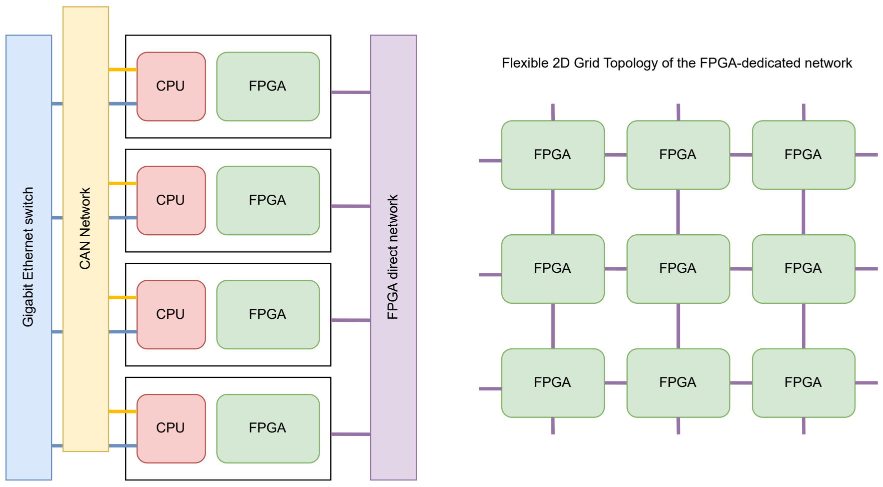


Figure 3.5: HyperFPGA network organization and FPGA dedicated network topology.

RJ45 CAT 8 cables to a switched Gigabit Ethernet network. Users interact with the system mainly through this network. The 4 board-to-board connectors and the PCB provide the infrastructure for a CAN network to join all processors. The CAN bus network is configured to use a set of headers to create a common network throughout the cluster. At the same time, these connectors allow developers to take advantage of the large quantity of GPIOs present in the FPGAs; 18 high-performance and 6 high-density pairs are available on each connector, along with the GTHs. Each high-performance and high-density pair provides additional throughput of 1.2 Gbps and 500 Mbps at double data rate (DDR), respectively. The maximum throughput is 388 Gbps for each board, depending on SoM resources.

Owing to the square shape of the PCBs and the coplanar board-to-board connectors on their sides, the physical topology is that of a 2D mesh network. However, the topology can be manipulated by using interface boards and cables. Using such an interface board, it is possible to transform a 2D mesh into a torus by interconnecting the two pairs of opposite edges.

Figure 3.6 shows the cluster in its current state. It consists of 16 nodes placed on a solid structure that keeps the connectors from detaching or breaking. All nodes are connected by an Ethernet cable for management and a power cable.

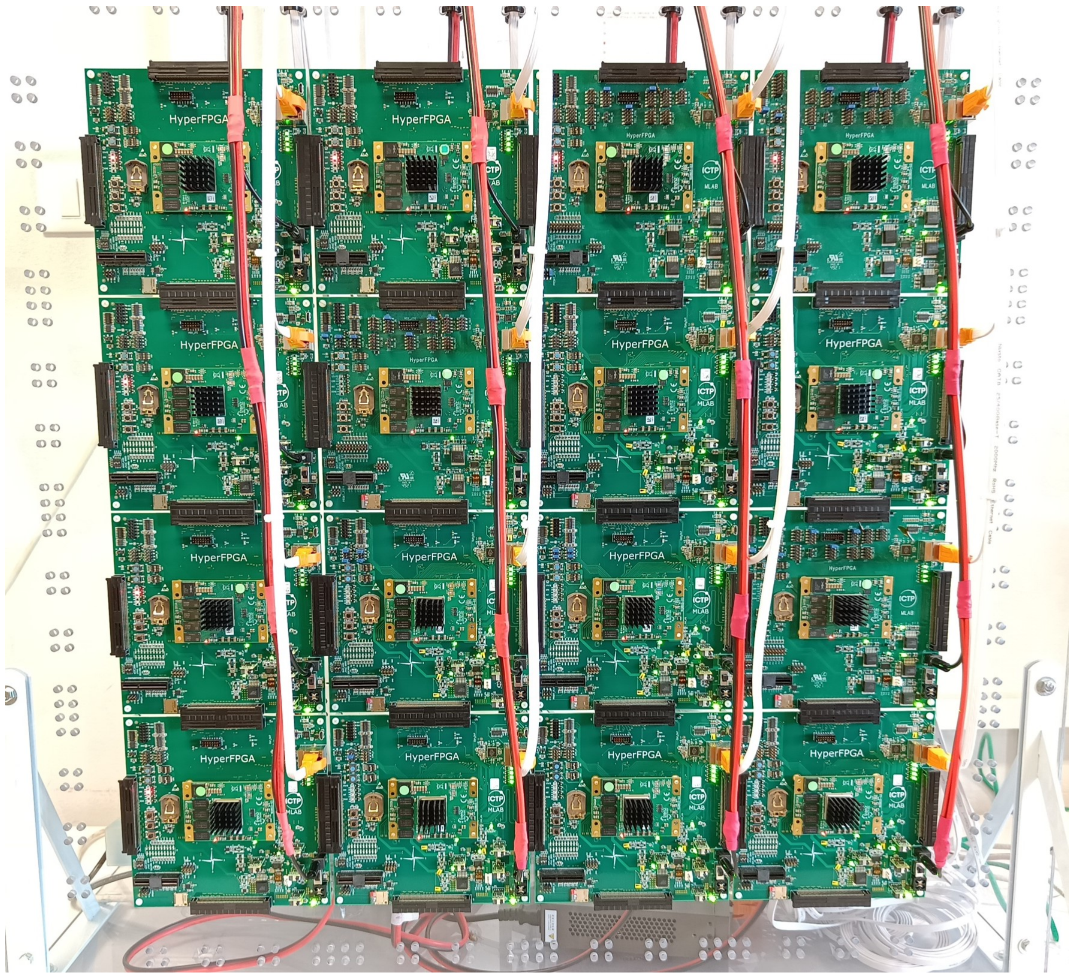


Figure 3.6: HyperFPGA cluster in its current state, consisting of 16 nodes.

3.2 Summary

In this chapter, the hardware design of the HyperFPGA prototype is described, with the aim of creating a versatile platform for experimental heterogeneous computing following the design proposal presented in the section 2.4. The definition of the carrier board is discussed along with the physical constraints that satisfy the envisioned characteristics. Being flexibility one of the most important aspects, custom hardware was developed and manufactured. The primary hardware choice is the SoC-FPGA due to its diverse capabilities that, combined with an open and standardized SoM layout, enhances modularity and adaptability. In addition, the design simplifies the complexity of the hardware and focuses on interconnections, safety, and power monitoring.

The HyperFPGA node is presented as a square PCB with high-speed connectors on each side that can be interconnected to form clusters. It exploits the abundance and variety of I/O present in the FPGA by offering them through such connectors.

In addition, it incorporates segmented power domains for power profiling. Users can extract information about how the implementation of their problem impacts both hardware and software by evaluating the performance of the corresponding power domains. This allows for fine-tuning algorithms for improved performance and also for developing better power estimation tools. The carrier board employs high-speed connectors to interconnect adjacent boards, establishing a flexible 2D mesh network topology that can be adapted to a torus.

This chapter lays the foundation for the HyperFPGA prototype, emphasizing modularity and adaptability. With a theoretical maximum throughput of 388 Gbps, the HyperFPGA board shows that it is suited for high-performance loads. These along with the other hardware choices and design principles underscore the platform's potential for experimentation in heterogeneous computing architectures. With respect to the state-of-the-art, the HyperFPGA carrier board offers a unique level of flexibility and introspection, which spans from the use of general-purpose connectors and independent monitoring of the power domains.

The BSP, schematics, and PCB files required to modify or reproduce the HyperFPGA board are openly available at the following link:

`https://gitlab.com/ictp-mlab/hyperfpga-hw`

Chapter 4

HyperFPGA software and network description

A useful analogy when thinking about distributed systems is to imagine a living organism, for example, a human. Hardware takes the place of organs that have specific functions. Each of the organs ingests and produces an output that allows the final organism to thrive and live. Nevertheless, organs in themselves cannot perform all duties if isolated. Through a standard way of communicating, organs can interact to perform more complex tasks in collaboration. The network becomes both the circulatory system and the nervous system. Finally, the software in conjunction with the application describes how each of the organs collaborates through the nervous system to achieve a collective goal, for example, to carry out this research and write this thesis.

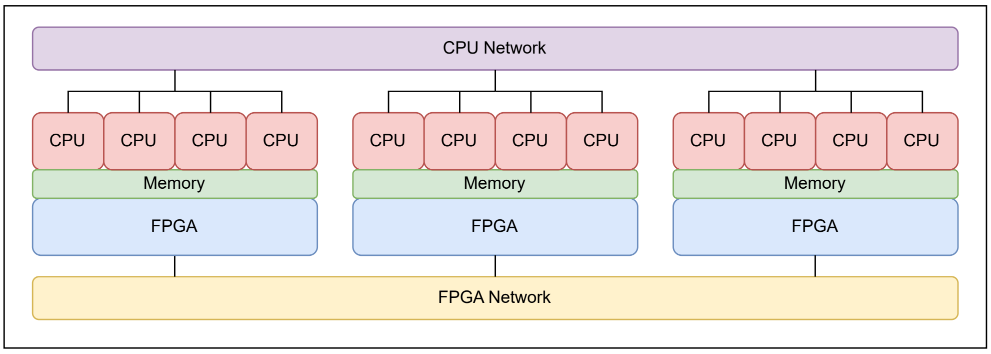


Figure 4.1: HyperFPGA network implementation of a cluster of SoC-FPGAs with distributed memory and dedicated FPGA-network (ARM quad-CPUs with interconnected FPGAs).

From the description of the HyperFPGA design in chapter 2, it was determined that the best network implementation would be one that would allow FPGAs to interconnect directly. From Figure 2.2 it can be recalled that the nature of SoC-FPGAs forces a classification to be type A or C. Type A refers to a cluster of distributed memory where the FPGAs are attached to CPUs and these act as gateways to the network, wasting the potential of the FPGA for high throughput and low latency applications. Type C networks allow for taking advantage of FPGA capabilities by providing an additional FPGA-dedicated network. In this way, FPGAs can collaborate without having to go through the CPU, which is in fact the case for the HyperFPGA, as shown in Figure 4.1.

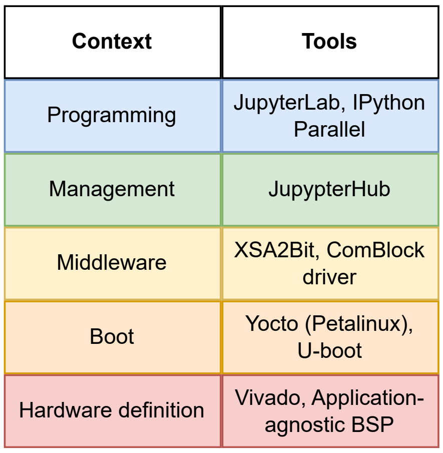
In the case of the HyperFPGA, two major networks were designed to provide maximum interconnection. One network involves all CPUs, and its main purpose is providing a gateway for configuration and control of the system nodes. The second major network is made up of all FPGAs in the system. There is a third network, the CAN bus, that connects all CPUs. This network has no specific purpose as of now, but is available for users to experiment with.

In addition to the networks, a software environment was developed to allow users to interact with the system individually and collectively. This environment provides safe abstractions for both experienced and novice users, to carry out their experiments. When considering that these experiments may be extremely different in their programming paradigm, synchronization protocols, and data structures, among other aspects, the environment must be flexible enough.

The pursuit of flexibility may become a slippery slope, where designers decide to abandon any boundary and provide just the bare minimum to the user. Although this might offer the highest degree of flexibility, users are forced to become experts in the platform, taking the focus away from the actual experiment. For this reason, the HyperFPGA software environment is made up of various modular components that can be swapped by the user according to the experimental requirements.

4.1 Software and CPU network overview

The software environment not only provides means of interaction, but must also facilitate managing, monitoring, and developing applications for the cluster. This includes boot files, the OS, the application environment, and APIs, among others. More often than not, the specific characteristics of each reconfigurable supercomputing system push for custom tools. This was the case for RAPTOR [11], which consisted of the development of a complete software stack, including a graphical user interface. However, similar solutions tend to restrict users to paradigms related to the architecture or target application. To escape this, the HyperFPGA provides the minimum interfaces to guarantee the safe operation of the cluster, such as the CPU-FPGA communication channel and its high-level abstraction. Additionally, the programming paradigm remains completely at the discretion of the user given that the base services provide configuration, monitoring, and safety measures. A completely programmable approach allows the system to act as a generic testbed for heterogeneous computing strategies.



Context	Tools
Programming	JupyterLab, IPython Parallel
Management	JupypterHub
Middleware	XSA2Bit, ComBlock driver
Boot	Yocto (Petalinux), U-boot
Hardware definition	Vivado, Application-agnostic BSP

Figure 4.2: HyperFPGA software stack showing the tools and their application context.

The HyperFPGA software stack, shown in Figure 4.2, is open and modular, allowing users to adapt it according to their experiments. Custom tools were developed in the hardware definition, boot, and middleware layers to provide an open standard interface that allows integrating already existing tools in the upper layers. From the ground up, the stack begins with hardware definition tools and facilities that configure the peripherals on the carrier board and SoM for their usage from the SoC-FPGA. This layer is also accessible for users to implement their application-specific FBs. On top of this, the boot tools implement the required security checks and hardware configurations for an OS to run on the ARM CPUs. To interface the OS and user space with the hardware, the middleware consists of a custom script that generates the required files to identify the user-created hardware in the FPGA, and a custom ComBlock

Linux driver, which provides a safe interface with the FPGA.

A management layer consisting of *JupyterHub* facilitates all administrative tasks. And at the top of the stack, the programming context consists of *JupyterLab* along with *IPython Parallel* which provides the tools for CPU cluster management. The whole stack depends on open tools, wherever this was an option, which are easy to adapt and port for other vendors or clusters.

The CPUs are interconnected via GbE over a switch. This allows the system to connect to the Internet and provide access to users and administrators. Additionally, this network can also be used to distribute tasks in a cluster of processors, enhancing the possibilities of experiments. Furthermore, the CAN network provides a deterministic link between processors that could be used for synchronization and other time-critical tasks without occupying FPGA resources.

4.1.1 Hardware definition

The hardware definition layer is critical given that it provides all the information on the peripherals and resources present on the board for the SoC-FPGA. In the case of the HyperFPGA carrier board and the Trenz SoM, information on SDRAM, UART ports, I2C, Ethernet, CAN, PCIe, micro SD, and GPIOs must be provided. Given that the selected SoC-FPGA is locked in the Xilinx environment, the configuration files must be available in a way that Vivado is capable of processing. For compatibility, the files are provided in the shape of a BSP, greatly simplifying the setup of the platform.

The BSP should remain fixed for most of the time unless there are hardware changes in the board or the SoM. At this level, no bitstream is provided. Although most of these configurations regard the CPU, all FPGA developments must be made considering the BSP to avoid possible incompatibilities with the already configured CPU. The FPGA hardware definition, which corresponds to the definition of the FBs, is also dependent on Vivado for the bitstream generation for the selected SoMS. This layer is available to users through the programming layer by using the OS drivers.

4.1.2 Boot

Establishing the initial state and the correct configuration is crucial for proper operation of any system. Additionally, the system should be capable of loading the operating system and performing all necessary services. The system's ability to consistently perform these essential operations with a high success rate is a measure of its robustness.

Typically, SoMs rely on static memory to host parts of the boot files, including the bitstream. This is suitable for a system with a fixed FPGA configuration, which is in stark contrast to the HyperFPGA philosophy. Consistent with the fact that the purpose of the HyperFPGA is to take full advantage of the ARM A53 quad core, a custom Linux-based OS was selected for which the files required for the initialization of the platform may quickly exceed the available flash memory on the SoM. To overcome this obstacle, the micro SD card port on the carrier provides hundreds of gigabytes to host not only booting files, but also the file system and user space, while being cheap and easy to replace.

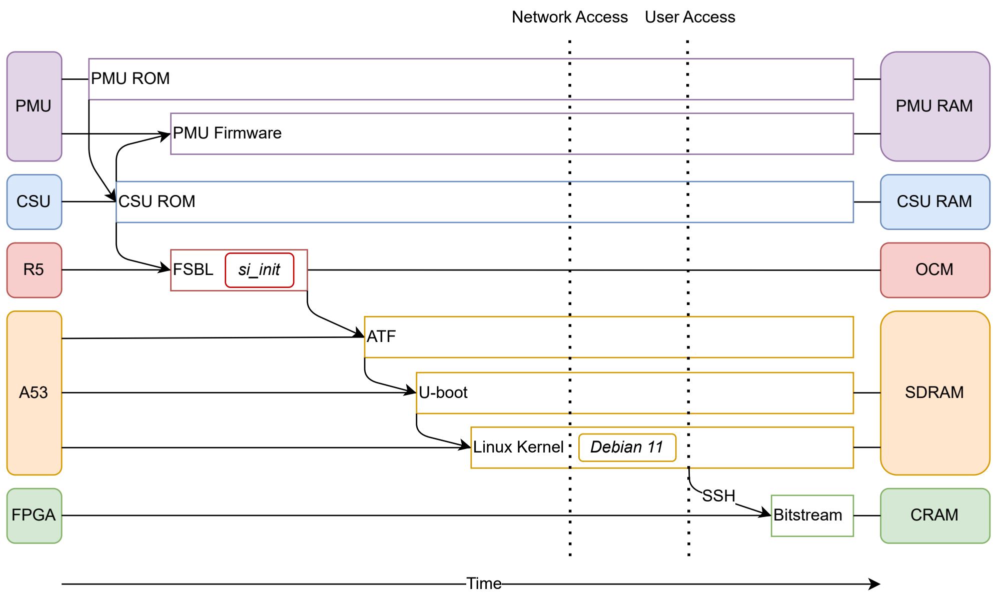


Figure 4.3: HyperFPGA boot sequence showing hand-off points. The custom FSBL ARM R5 processors configures the SoM oscillators to ensure the GTHs are getting a clock signal before usage. A *Debian 11* minimum OS is the last part of the standard booting process. The FPGA is left without configuration, waiting for the user bitstream.

Figure 4.3 shows the HyperFPGA boot process. Initially, the Platform Management Unit (PMU) and Configuration Security Unit (CSU) processors boot with the default Petalinux binaries, which are based on the open-source project named Yocto. If the previous step is successful, the CSU loads the boot header to configure the ARM R5 with the First Stage Boot Loader (FSBL). The R5's FSBL is the first custom component in the boot process. This firmware configures the peripherals on the carrier and ensures that the GTHs on the SoM have a valid clock before continuing. This acts as a failsafe to avoid booting in a state where the GTHs are unusable. Once this operation is completed successfully, execution is handed to the ARM Trusted Firmware (ATF). The ATF is the first firmware to run on the ARM 53 processors. It allows using the ARM Exception Model which is essential for deploying any OS on the SoC. With the ATF running, the U-boot loads the custom Linux kernel. Finally, the OS, a *Debian 11*, boots with network access for user interaction. The FPGA is left with the boot configuration until the user applies a bitstream in the programming layer.

The boot sequence can be implemented in several ways, depending on the desired level of robustness, such as hosting a minimal boot on the SoC-FPGA flash and using the SD card as a backup in the event of corruption. Furthermore, a file transfer protocol (FTP) server could be used to host a copy of the OS for additional resilience in hard environments or when the OS is part of the user application. These implementations have already been proven to be successful on other platforms [168, 169]. Nevertheless, given that in the case of the HyperFPGA the environment is controlled and always accessible. Board initialization is only completed when the user can interact with the platform remotely, and it makes little sense to give control in any previous step.

4.1.3 Middleware

The HyperFPGA strives for compatibility and portability and this can only be achieved by using standard and appropriately documented interfaces. Multiple standards are already available in the tool stack for interfacing layers.

One of the most useful is the device tree which provides Linux-based OSs with a dynamic description of hardware and how to interact with it so that the OS does not need to hardcode details of the platform. This is particularly useful, given that the user-defined hardware in the FPGA is implemented in runtime. Linux allows partial changes to the device tree through overlays.

Here is where a custom in-house developed Python script called XSA2Bit becomes essential. It translates the Vivado XSA (Xilinx Support Archive) file that contains all the information needed to build a platform for a user's target device into a device tree overlay. This script makes it possible to update the OS with the new FBs in the FPGA in a transparent way without major user intervention. Furthermore, it automatically instantiates ComBlock devices that interface the FPGA with the CPU for use with custom kernel drivers. The advantages of relying on the ComBlock are discussed in detail in [170] but they can be summarized in the abstraction of the SoC-FPGA bus resulting in portability and the separation of the programming of CPUs and FPGA, since these two subsystems only need to agree on the logical utilization of the ComBlock. Additionally, the usage of a kernel driver protects the memory and allows users to build drivers using only common file operations (read, write, and seek).

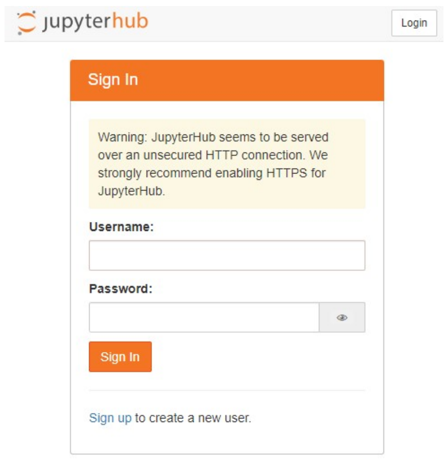
4.1.4 Management

The first layer of interaction of the user is through the management services of the platform. This contains authentication, authorization, and accounting services. To simplify the development of the platform, we decided to rely on *JupyterHub*. Given that all nodes already run a custom Linux-based OS on ARM cores, the implementation was trivial.

JupyterHub provides several authentication methods and its open-source nature simplifies customization. Currently, the system is deployed on a local network without external access using the *native Linux authenticator*. Users can request to be registered, and the administrator can enable the account without storing sensitive information from the user.

Whenever a user accesses the URL of the *JupyterHub* server from any browser, the user is greeted with a login page, shown in Figure 4.4. On this screen, the user can login using the selected username and password or request an account. When a new user account is requested, the intervention of an administrator is required. A dashboard provides the administrator with all the tools to create the new account and assign the desired privileges.

As the name suggests, *JupyterHub* is based on a centralized management server that greatly simplifies the support hardware required for the cluster by constraining all management effort to this server without polluting the nodes. *JupyterHub* relies on spawners to enable user interaction with the system through *JupyterLab* sessions. By modifying the behavior of the spawner, it is possible to run different configurations to customize the user interaction within the system.



Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

Username:

Password:

Sign up to create a new user.

Figure 4.4: *JupyterHub* greeting screen, here users can log into their account or request the creation of a new account.

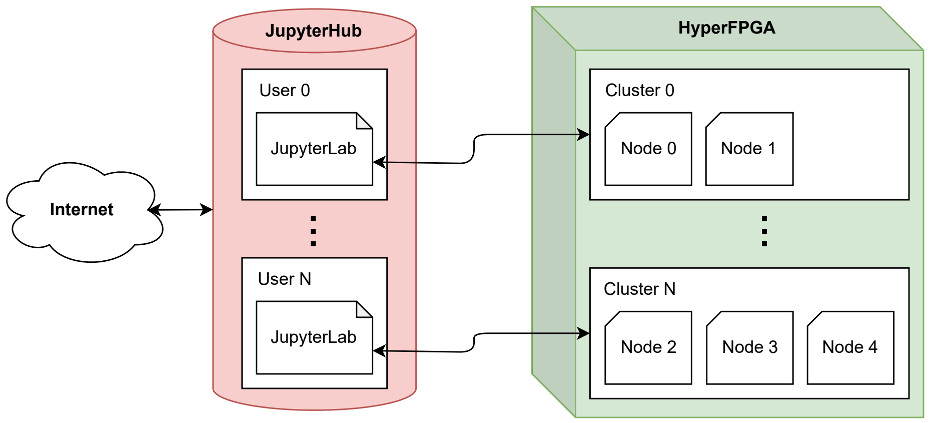


Figure 4.5: User interaction block diagram with *JupyterHub* centralized management and remote clusters instantiated using HyperFPGA nodes; spawner initiates local *JupyterLab* session on the hub where notebooks can be created, modified, and executed.

Figure 4.5 shows the diagram corresponding to the cluster implementation. It shows the hub as the only gateway for interaction with the cluster but also as a centralized file system for the user's configuration files, experiment results, and applications. Management tasks are mainly limited to this part of the system and consist of enabling user accounts, upgrading system-wide libraries, securing data storage, and assigning nodes to users which is done manually at the moment. For this implementation, the spawner starts a *JupyterLab* session on the hub server, and communication is forwarded to a given cluster of nodes through Ethernet.

The administrator assigns nodes to a given user via a JSON (JavaScript Object Notation) conveniently named *nodes.json*. The JSON format is an open standard for data interchange in human-readable text that is widely supported, making it easy to understand and modify when necessary. This file provides all relevant information about the hardware of the nodes for the correct development and implementation of an application.

Listing 4.1 shows the structure of the *nodes.json* file. It defines an array of nodes by describing their parameters, such as `hostname`, `ip`, and position in the 2D grid. The most relevant part of the `node` is the `fpga` entry, which indicates the FPGA `model` present at the node, its `state`, and the `firmware` used to configure it. These last two parameters are dynamically updated whenever the user configures the FPGA with a bitstream that matches the `model`. Finally, the `comblock` entry provides insight into the available ComBlock resources by listing the device files.

Listing 4.1: JSON cluster description example

```json
{
    "nodes":[{
            "hostname":"hyperfpga-4ge21-1-1",
            "ip":"192.168.0.7",
            "x":1,
            "y":1,
            "fpga":{
                "model":"4ge21",
                "state":"unknown",
                "firmware":""
            },
            "comblock":{
                "devs":[]
            }
}
```

Furthermore, *JupyterHub* flexibility has proven to be beneficial for user interaction on individual boards. By implementing the *SSHSpawner* [171], users are further restricted to a specific node of the cluster; this can be useful for teaching activities [172]. The *SSHSpawner* takes advantage of the network by initiating a remote *JupyterLab* session on a node and then forwarding it to the hub server as shown in Figure 4.6. In both cases, the gateway to the hardware is a common server that can easily implement all safety measures that require minimal programming skills.

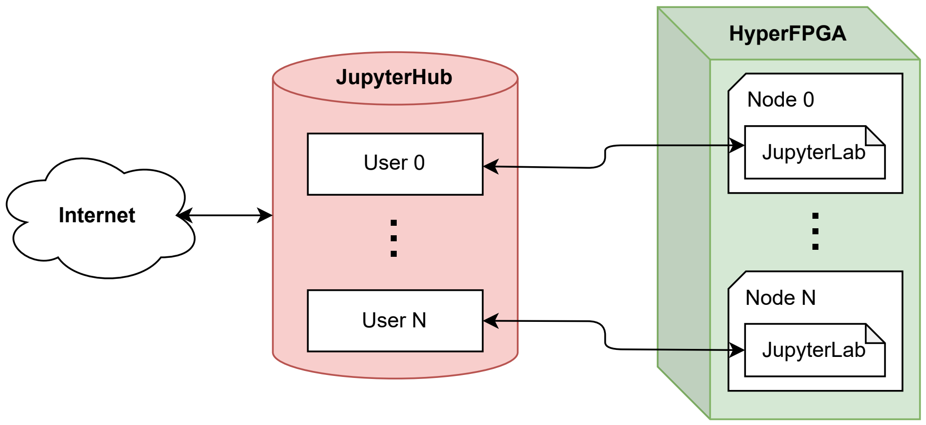


Figure 4.6: Block diagram of the infrastructure configured for user interaction with individual nodes through *SSHSpawner*. Users are authenticated in the hub and the spawner initiates a remote session that is forwarded to the hub for user interaction.

4.1.5 Programming

Computing in general is a two-part task. In most problems, the configuration may be just the beginning, with the real challenge being the programming of the newly created cluster. At this point, the architecture of the cluster may be very different for each problem, but a flexible framework adapts accordingly to allow users to implement their applications without having to make major modifications to the underlying layers.

Figure 4.7 shows the user workflow and tools required to develop and implement an application on the cluster. For starters, HDL code can be developed in any language or tool that outputs VHDL or Verilog, which are the only two languages supported for hardware synthesis in the case of Vivado [26]. The methodology of SoC-FPGA implementation follows the princi-

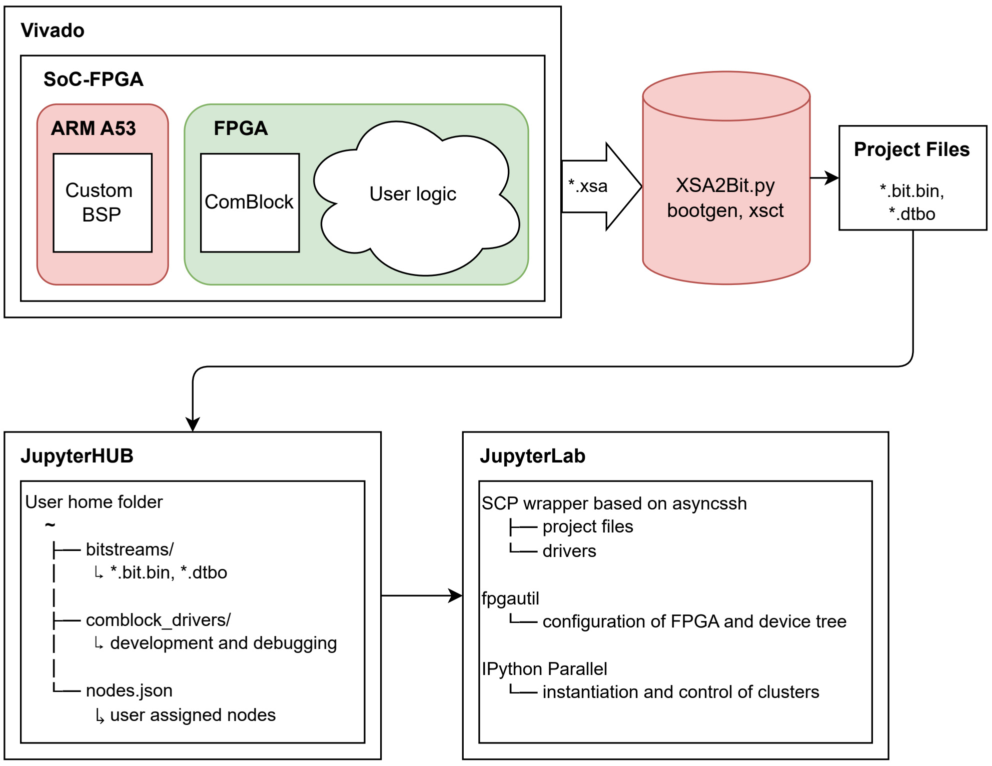


Figure 4.7: HyperFPGA application development workflow.

ples described in [173], where the communication block (ComBlock) [174] is used as a bridge between user reconfigurable logic and the CPU, abstracting all details of the interconnect. By abstracting the interface details, the ComBlock provides a compatibility layer consisting of registers, FIFOs, and a dual-port RAM allowing to facilitate design portability.

The HyperFPGA platform configuration is provided in a BSP format, compatible with Vivado. Once the user design has been integrated and synthesized with the hardware platform, it can be exported to be processed by the XSA2Bit Python script. The XSA2Bit helps to compile the bitstream and generate the device tree overlay for compatible devices. With the project files compiled, users can take advantage of a *Jupyter* environment, which includes *JupyterHub* [175] for management and *JupyterLab* for application development.

Once users are logged into the system in the cluster scheme, a *JupyterHub* greeting page shows a number of options. Users can interact through a shell terminal or *JupyterLab* notebooks.

Any modification made to the system is limited to the user account.

Within the user home folder, the *nodes.json* file contains the information of the cluster nodes available to the user. This will be used to implement and interact with the CPU clusters to configure and monitor the computations in the FPGAs. In addition, two folders are created within the user account.

The first folder is named `bitstreams`. It contains the bitstreams and device tree overlays that target the specified FPGAs on the available nodes. During configuration, the bitstream and device tree overlays are checked to match the targeted FPGA model. If valid, the CPUs implement the bitstream in the FPGA and apply the overlay updating the Linux device tree.

The second folder, named `comblock_drivers`, contains the Linux kernel drivers that the user requires to interact with the FPGA of the ARM A53 quad-core CPU via the ComBlock. Two different variants of the drivers are available, one for normal usage and one for debugging, which allows inspection of the interaction between the kernel and the ComBlock. The drivers are capable of handling multiple ComBlocks with different configurations. After a device tree overlay is successfully applied to the OS, the kernel driver creates the device files for normal usage, effectively exposing the ComBlocks in the FPGA to the user space.

This application framework is based on *IPython Parallel* [176]. It provides functions that facilitate the instantiation and management of CPU clusters. With the description file of the nodes available to the user, it is possible to create such clusters. The interaction with the cluster is carried out using the methods provided with the *IPython* environment or *MPI4PY* [177]. Both tools provide useful functions for sharing Python objects and performing computations, such as `broadcast`, `barrier`, and `map`. These functions combined with the ComBlock drivers allow for a heterogeneous computing flow where the CPUs and FPGAs can cooperate.

4.2 FPGA dedicated network

To enable the real potential of FPGAs massive parallelism, they must be provided with a dedicated network with low-latency and high performance. In the HyperFPGA this is achieved by interfacing the I/Os directly. As mentioned in the previous chapter, the combination of GTHs, HPs, and HDs allows for a 380 Gbps link between boards.

Given that the HyperFPGA is intended as an experimental playground, the methods of communication between FPGAs are highly dependent on the user application. However, the same modular approach can be implemented at this level. By leveraging already existing modules, users can implement portable implementations. As described in the experiments carried out with *Novo* [110] and *Novo G#* [111], depending on the targeted optimizations, area, performance, or latency, one could prefer a network strategy over others.

In terms of modularity, the HPs and HDs are the most simple to deal with. Given that these are common I/Os, the user only has to define the electrical standard and a direct connection can be established. These were tested using a FIFO on both sides and routing the FIFO control and data signals directly to the I/Os. A performance of 600 Mbps per differential lane was achieved when working with HP I/Os using the HyperFPGA and Trenz SoM.

The case for the GTHs is quite different, given that they require a more complex configuration, which tends to be vendor and family specific. This overhead is acceptable considering the potential throughput on the order of Gbps. Despite the complexity and restrictiveness in its use for portable applications, they are essential to any high performance system. For this reason, efforts are present in developing a universal GTH controller such as *Kyokko* [167].

Regarding GPIOs, some experiments were performed to test the feasibility of physical-level interfaces based on time-division multiplexing (TDM) [178]. These types of interface offer the advantage of fixed-length frames in which the channel is owned by different entities for predetermined periods of time, making the link deterministic at the cost of agility.

In addition to this, a proposed protocol was presented in [179]. It consists of a distributed memory approach in which all nodes are assigned a memory range in a global memory space. This effectively transforms all ports in FBs or CEs to memory locations which can be written and read from. Inside each node, local resource agents manage communication with other nodes by means of packets. These packets are formed of various layers, depending on the type of packet. Three types of packets are considered: command, raw data, and universal direct memory access (UDMA).

The command packet provides a way of communicating short urgent messages with codes whose meaning is predefined for start, stop, restart, abort, etc. Data are transported using the raw data packet type, which provides packet sequencing, checksum, and addressing, among others. Finally, the UDMA packet provides the means to explicitly declare data movements in remote nodes to remote nodes. This gives the flexibility to move data across the cluster in an arbitrary way.

4.3 Summary

The usability of any system depends not only on the potential qualities of the system but also on the capacity of users to interact with it. In the previous chapter 3 the hardware qualities of the HyperFPGA were described. To make use of these, tools were developed that provide facilities for management, monitoring, and development. These tools allow users to take advantage of the two major networks that connect CPUs and FPGAs.

The HyperFPGA software stack is designed to be flexible and modular, so that users can develop applications that meet their specific needs. It includes the hardware definition, boot, management platform, and programming environment. The importance of proper configuration and the boot sequence is emphasized, highlighting the use of micro-SD cards for flexibility and robustness. The rest of the software ecosystem relies on *JupyterHub* for user management and interaction. The software environment allows users to interact with the system nodes individually and collectively, accommodating different programming paradigms and use cases. By providing drivers for the redefinition of the hardware present in the FPGA and for their interaction from the user space, developers can modify the platform to accommodate their application intervening in different layers while maintaining compatibility with the system.

Following an open-source approach, the Petalinux project, kernel drivers, ComBlock, and XSA2Bit sources are provided in the following links for free use:

`https://gitlab.com/ictp-mlab/core-comblock`

`https://gitlab.com/ictp-mlab/hyperfpga-linux`

Chapter 5

Application: N-Queens problem

In the previous chapters, the characteristics of the HyperFPGA were described in detail. As stated in the design proposal in Section 2.4, the main purpose of the system is to provide an ecosystem for heterogeneous experimental HPC architectures. From the hardware description in chapter 3 the heterogeneity of the cluster is shown in the form of the SoM that houses diverse systems in a single SoC. Later, in Chapter 4, the way to utilize these resources is presented. Being a distributed system, strong emphasis is placed on communication and synchronization. And being an experimental platform, special care was taken for management and safety.

The natural next step, once the platform and software ecosystem is ready and running, is to develop an application to test the HyperFPGA cluster. There are some requirements that any application must meet before it can be considered fit for an HPC platform. The application must:

- Exceed the resources available in a single CE.
- Be scalable.
- Be parallelizable and distributable.

Following these requirements, it was decided that the N-Queens problem was a good choice to validate the HyperFPGA due to the combinatorial nature of the problem according to the dwarf classification introduced in Table 1.1. The N-Queens problems has been traditionally used in the testing of HPC systems [180]. This problem is ideal for showing the capabilities of the hardware and software that build up the HyperFPGA given that no formula exists for the computation of the number of solutions for the placements of N non-attacking queens, and exhaustive search must be used to go over the whole solution space.

5.1 N-Queens problem

The Queens problem was first introduced in 1848 by Max Bezzel [181], a German chess composer. In its first iteration, the problem consisted of placing eight queens on a traditional eight by eight chessboard. The queens have to be positioned in such a way that they were not be able to attack each other.

It took some time before a generalization was proposed, as expected from mathematicians. The N-Queens problem in its general form has been extensively studied [182, 180, 183]. As the size of the chessboard (N) increases, the number of total solutions increases approximately with the factorial of N. Currently, there are three main variations of this problem.

- Find a single solution.
- Find the first solution in lexicographic order (most beautiful [184]).
- Find all possible solutions.

There are many ways of solving this problem, one of the most popular was proposed by Dijkstra [185] as an example of structured programming using back tracking. Naturally, several improvements have been made to the algorithm to reduce the complexity of the problem by taking advantage of the symmetries of some solutions. Until now, the total number of solutions has been confirmed for N = 27.

5.1.1 Implementation

This particular problem tests the scalability of a platform and has been solved in numerous ways. First, given that there is no mathematical equation or model, an exhaustive search is the best way forward. It is a brute-force approach that systematically lists all possible solutions to a problem and checks each for valid solutions that require huge computational power. Second, all possible solutions are independent, allowing massive parallelization. Furthermore, pruning techniques allow discarding solutions without completely checking them. Thus, the problem can be deconstructed into a series of sub-problems that branches out from valid partial solutions.

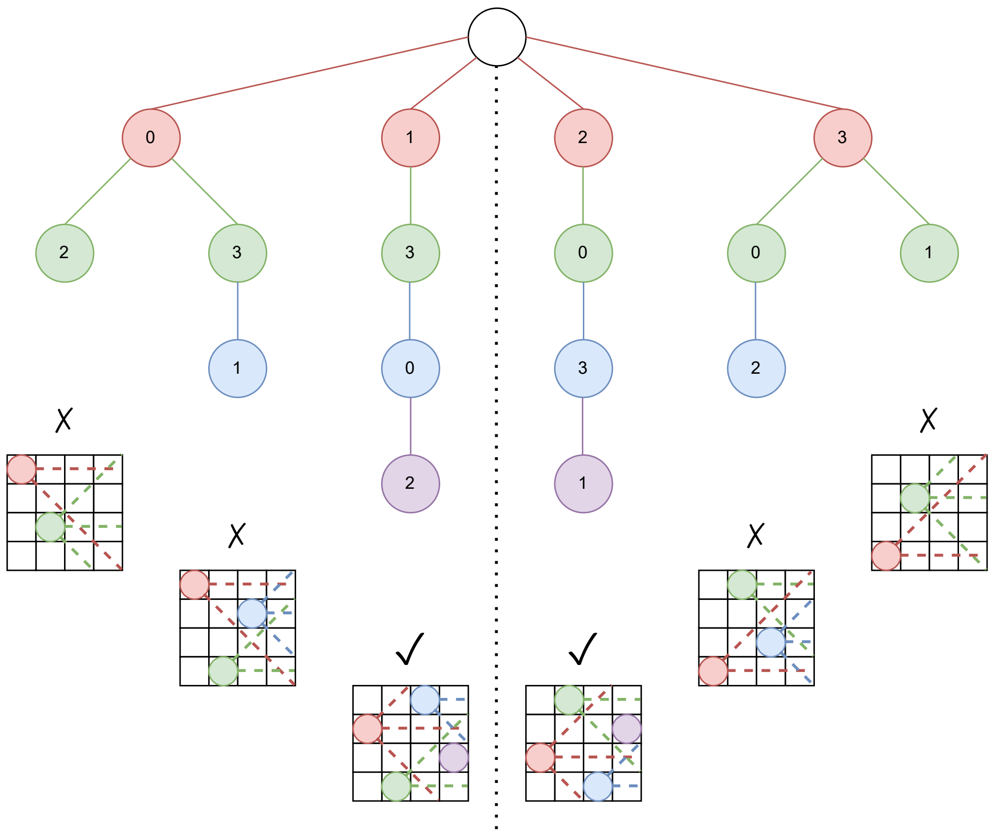


Figure 5.1: Example of the exhaustive search process using the backtracking algorithm in a 4-queens problem. Queen's placements are shown with circles and attacking lines in different color depending of the column.

A backtracking algorithm was used to optimize the implementation and reduce the computational time required to find all the solutions. Figure 5.1 shows how the problem branches whenever a queen is placed on the board. An advantage of applying backtracking is that it follows a depth-first search that allows parallelization by distributing branches to the cluster nodes. Furthermore, the space to explore is greatly reduced by pruning whenever a non-valid solution is found for a given partial solution, as discussed in [186].

5.1.1.1 Hardware

To solve the N-queens problem, it was decided that part of the solutions could be calculated on the JupyterHub. This consists of finding the valid positions for the first columns of the chessboard in the CPU, since this is the less computationally intensive part of the problem. Then, the partial solutions are distributed to the nodes to maximize efficiency. Inside the nodes, the backtracking algorithm was implemented using parameterized VHDL templates in modules called solvers.

Figure 5.2 shows the FPGA block diagram inside each node. By reading and writing to the ComBlock resources, the entire logic can be controlled. The ComBlock FIFO is used to store all partial solutions assigned to each node. A distributor reads from the ComBlock FIFO and copies the partial solutions to each solver in a round-robin manner. The number of solvers is restricted by the physical resources of the system and the number of partial solutions to be processed.

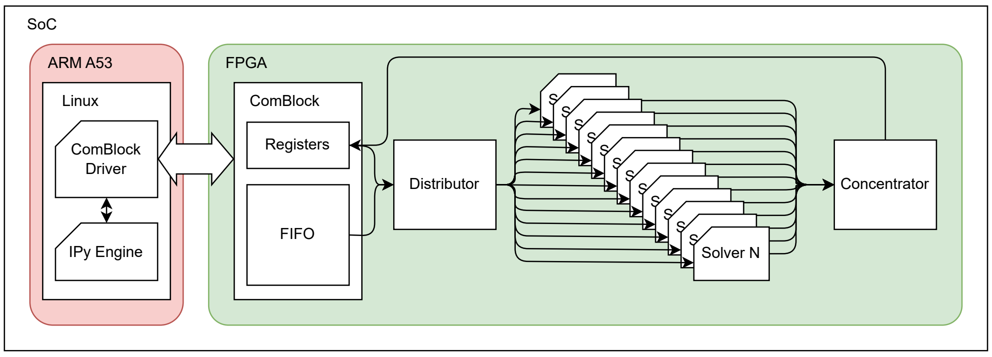


Figure 5.2: N-Queens problem block diagram showing the solvers, partial solutions distributor, partial results concentrator, and ComBlock.

The control logic inside each solver performs the search for valid positions in its column. If no valid position is found, the partial solution is discarded and the next is processed, as shown in Algorithm 1.

The block diagram of an individual solver can be seen in Figure 5.3. Inside each solver, the backtracking algorithm is implemented by unrolling the columns of the chessboard into N logic control blocks with its own input FIFO to minimize dead times. Each logic control block takes a partial solution from the preceding block and checks for valid solutions in its designated

Algorithm 1 K column solver control logic for an N by N chessboard.

```
Require: Previous placements $\neq NULL$
  for Proposed position $\leftarrow 0$ to $N-1$ do
      for $i \leftarrow 0$ to $K-1$ do
          $Counter \leftarrow 0$
          if Proposed position not in Previous placements($i$) horizontal AND diagonals then
              $Counter \leftarrow Counter + 1$
          end if
      end for
      if $Counter = K$ then
          Proposed position valid
      else
          Proposed position invalid
      end if
  end for
```

column. Any valid solution is concatenated to the partial solution and fed to the next block. When the last block finds a valid solution, the solver increments its solution counter. Once no more partial solutions are available for processing and the solver has completed its assigned computations, a `done` flag is raised. Finally, when the concentrator detects that all solvers have raised the `done` flag, it accumulates the values of all counters and updates the ComBlock registers.

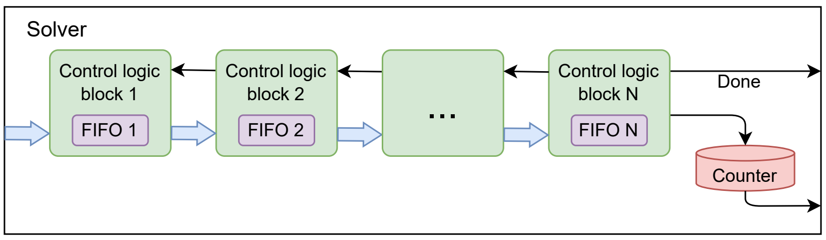


Figure 5.3: N-Queens solver internal block diagram showing N control logic blocks with their input FIFOs. The last control logic increments the solutions counter and generates the `Done` flag.

The FPGA part is completely developed in Vivado due to the proprietary restriction of the AMD MPSoC Ultrascale+. However, once a definition of the hardware is obtained, it is possible to escape vendor entrapment, as shown in Figure 4.7. To do so, the XSA2Bit custom Python script reads the hardware description and translates it into a device tree overlay to interface the custom logic with Linux through the ComBlock and its custom driver. The overlay is populated with the parameters of the instantiated ComBlock to pass the driver information, such as the availability of a memory resource, depth, and width. The ComBlock entry in the device overlay is shown in Listing 5.1.

```
queens_0_comblock_0: comblock@80020000 {
    clock-names      = "fifo_clk_i", "axil_aclk";
    clocks           = <&zynqmp_clk 71>, <&zynqmp_clk 71>;
    compatible       = "xlnx,comblock-2.0";
    reg              = <0x0 0x80020000 0x0 0x10000>;
    REGS_IN_ENA      = <1>;
    REGS_IN_DWIDTH   = <32>;
    REGS_IN_DEPTH    = <3>;
    REGS_OUT_ENA     = <1>;
    REGS_OUT_DWIDTH  = <32>;
    REGS_OUT_DEPTH   = <1>;
    DRAM_IO_ENA      = <0>;
    DRAM_IO_DWIDTH   = <16>;
    DRAM_IO_DEPTH    = <0>;
    FIFO_IN_ENA      = <0>;
    FIFO_IN_DWIDTH   = <16>;
    FIFO_IN_DEPTH    = <1024>;
    FIFO_OUT_ENA     = <1>;
    FIFO_OUT_DWIDTH  = <32>;
    FIFO_OUT_DEPTH   = <256>;
};
```

Listing 5.1: ComBlock device tree overlay entry.

This entry is part of a larger device tree overlay that is then compiled. With this last part, a user can proceed to the *Jupyter* environment to build the distributed application.

5.1.1.2 Software

The applications relies on the *IPython Parallel* framework [176] which provides communication methods for cluster management. This framework provides valuable tools that abstract the details of distributed computing communication. The first step is to create a *Python* object to describe the cluster. This cluster must first be defined by the system administrator in the *node.json* file.

By reading the JSON and cluster profile files, a user can access all the information required to instantiate a cluster, which is done with simple function calls within *JupyterLab*, see Listing 5.2. With the cluster object, users can interact in a `DirectView` or `LoadBalancedView`, both provided by the *IPython Parallel* framework. These views change the way communication with the nodes of the cluster is carried out. In the `DirectView`, each node can be addressed specifically, giving more control. On the other hand, `LoadBalancedView` relies on a process that automatically distributes a set of tasks or data to nodes, with the aim of making their overall processing more efficient. For the implementation of the N-Queens problem, `DirectView` is preferred, given that it allows fine-grained control of the cluster while maintaining parallelism.

```python
import ipyparallel as ipp
import json

with open('nodes.json') as nodes_file:
    nodes = json.load(nodes_file)['nodes']

engines = 1
cluster = ipp.Cluster(profile = "ssh", n = engines)
rc = cluster.start_and_connect_sync()
```

Listing 5.2: Creation of a cluster in JupyterHub.

Before operation, the cluster is programmed with the bitstream and the OS is updated with the corresponding device tree overlay. Once the overlay is applied successfully, the status of the nodes is updated with the firmware name and the ComBlock device files. At this stage, the system is ready for computing.

Calculating partial solutions greatly boosts parallelism in the case of the Queens problem; to do so, the hub performs the simple task of generating all partial solutions for a given number of columns that are stored in `presols_o`. These partial solutions are randomly shuffled and distributed to the cluster using the `map_async` function which is a higher-order function that takes another function and data, see Listing 5.3. The `map_async` implementation in *IPython Parallel* distributes a given function and the element of a list to the nodes of the defined cluster in parallel to then return the results of each node.

```python
from random import shuffle

rc.block = False

chunk_size = len(presols_o)//engines + 1
shuffle(presols_o)
ar = dview.map_async(calculate_solutions_fpga, presols_o)
```

Listing 5.3: Execution of the N-Queens.

The function that actually performs the computation by controlling the solvers on the FPGA was conveniently named `calculate_solutions_fpga`. This function interacts with the solvers through the ComBlock device files using simple read and write operations. The advantage of using these operations is that since they are pervasive and standard, they are supported natively in Python and, thus, are easy to integrate with any Python package. In particular, one of the most useful tools are context managers that provide a safe environment to interact with external sources. Furthermore, the *IPython Parallel* framework provides monitoring tools, given that the FPGA does all the heavy lifting, the CPU can monitor without interfering with any critical task by using the `publish_data`; see Listing 5.4.

```python
def calculate_solutions_fpga(partial_solutions):
    with ExitStack() as context_manager:
        reg_o  = context_manager.enter_context(open_dev_file(f"/dev/
    ComBlock_0_regs_o", os.O_WRONLY, "bw"))
        reg_i  = context_manager.enter_context(open_dev_file(f"/dev/
    ComBlock_0_regs_i", os.O_RDONLY, "br"))
        fifo_o = context_manager.enter_context(open_dev_file(f"/dev/
    ComBlock_0_fifo_o", os.O_WRONLY, "bw"))
        reg_o.write((1).to_bytes(4, 'little'))              # write reg0 (reset
    high)
        fifo_o.seek(-1)                                     # empty the fifo
        last_sol = 0
        for i, partial_solution in enumerate(partial_solutions):
            try:
                fifo_o.write((partial_solution).to_bytes(4, 'little'))
                last_sol += 1
            except:
                break
        reg_o.seek(0)
        reg_o.write((0).to_bytes(4, 'little'))              # write reg0 (reset low)

        i = 0
        reg_i.seek(0)
        done = int.from_bytes(reg_i.read(4), 'little')
        while(not done):                                    # read reg0 (done)
            time.sleep(60)
            i += 1
            # read solution counter low and high words
            sol_l = int.from_bytes(reg_i.read(4), 'little')
            sol_h = int.from_bytes(reg_i.read(4), 'little')
            sol = (sol_h << 32) + sol_l
            reg_i.seek(0)
            done = int.from_bytes(reg_i.read(4), 'little')
            publish_data(dict(sol=sol, i=i))                # to monitoring task
            while fifo_o.tell() < 1024 and last_sol < len(partial_solutions):
                fifo_o.write((partial_solutions[last_sol]).to_bytes(4, 'little'))
                last_sol += 1
        sol_l = int.from_bytes(reg_i.read(4), 'little') # read low word
        sol_h = int.from_bytes(reg_i.read(4), 'little') # read high word
    return (sol_h << 32) + sol_l, i
```

Listing 5.4: CPU function that interacts with the solvers on the FPGA to calculate the solutions.

When the application is running, each of the nodes will publish its progress in *JupyterLab*. This is particularly useful for examining the computation and assessing the parallelization strategy during execution. Figure 5.4 shows the progress of 4 nodes, called engines, in the context of *IPython Parallel*. This corresponds to one of the iterations of the Queens problem. All engines remain operational throughout execution, showing that there is a high level of efficiency.

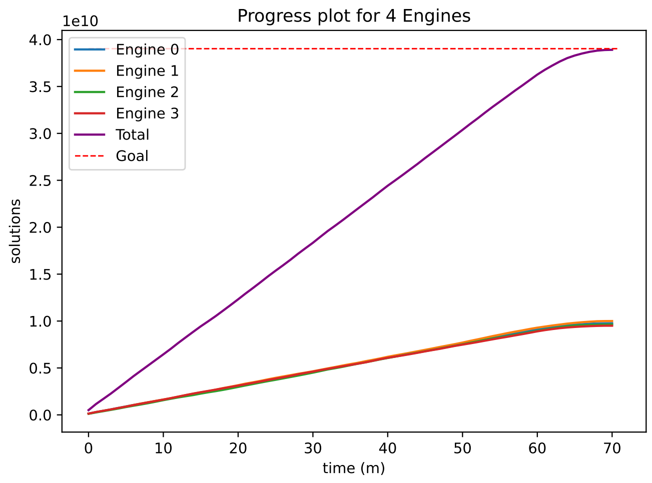


Figure 5.4: 20-Queens problem monitoring chart showing the progress of the individual counter of a 4-node cluster with the accumulated solutions and the expected total as the goal (39,029,188,884).

5.1.2 Results

During the algorithm development phase, the entire system was tested for the first time on a Zedboard [187]. The results obtained give us insight into how the problem grows. Figure 5.5 shows how execution time increases with the number of queens in a factorial way. This plot was drawn using a single solver at 100 MHz.

It was expected that, when switching from Zedboard to the HyperFPGA cluster, as the available resources for the problem increased, the execution time would decrease. In this case, the size of the chessboard (N) was fixed to 20 and up to 4 nodes were used. Figure 5.6 shows the platform gains when increasing the number of nodes. The execution time was measured multiple times to obtain the mean value and standard deviation for each case. Given that the computation time depends on the length of the search path for each partial solution, these were randomly assigned in each test to minimize the total execution time. As expected, when partial solutions are distributed to more nodes, the total time decreases, confirming that the platform allows one to exploit the algorithm's parallelization.

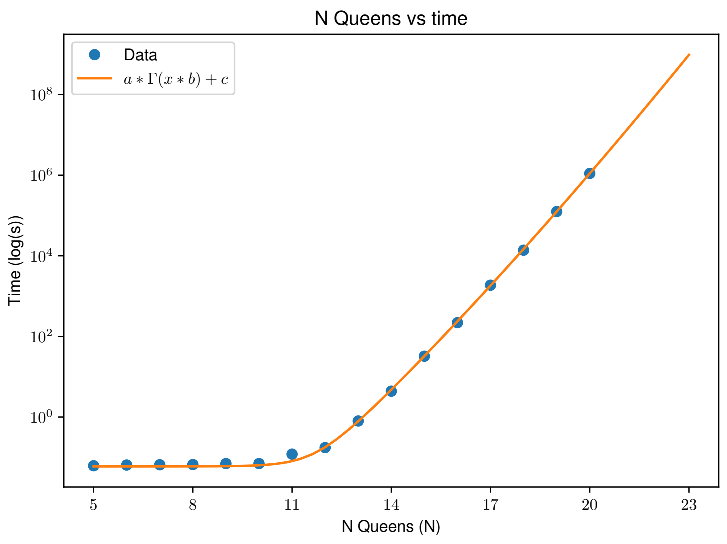


Figure 5.5: N-Queens problem time growth with real values from Zedboard experiments using a single solver at 100 MHz. The values were fitted with a factorial model confirming how the problem grows with N.

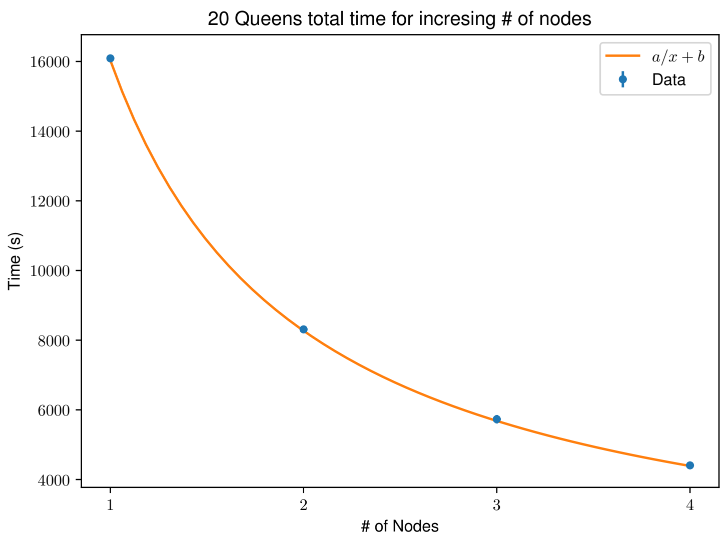


Figure 5.6: N-Queens problem speedup for a board size of $N = 20$ and different numbers of nodes, from 1 to 4. A model was used allow predicting the speed up for further scaling.

Another important factor is the number of solvers that fit into a determined FPGA. For this particular test, using the Ultrascale+ MPSoC ZU4EG, 23 solvers at 300 MHz were placed at each cluster node. By combining the information from both experiments, Figure 5.5 and Figure 5.6, for $N = 20$ a speed up of 251x was obtained by implementing the algorithm on 4 nodes of the HyperFPGA (276 equivalent solvers) versus a single solver on a Zedboard.

5.2 Summary

This chapter discusses a test application developed on the HyperFPGA system. The N-Queens problem was implemented since it is a complex combinatorial problem that grows in a factorial way. The implementation involved a backtracking algorithm implemented in VHDL on the HyperFPGA nodes, with the *IPython Parallel* framework used for cluster management. The platform demonstrated its capability to solve complex problems in a distributed manner, achieving a speed up of 251x when comparing the results obtained with a single Zedboard.

The implementation for the 20-Queens problem is available openly at the following link: `https://gitlab.com/ictp-mlab/n-queens`

Chapter 6

Conclusions

In this work, the HyperFPGA cluster is presented, which is a hardware and software platform for experimental heterogeneous high-performance reconfigurable computing architectures. The need for novel computing architectures has arisen from the current unsustainable trend of supercomputers that consume as much energy as small towns. This trend is impossible to support and highlights major issues of the current approach based on CPUs and GPUs. Meanwhile, FPGA-based heterogeneous platforms have shown significant improvements in performance and energy consumption compared to their CPU or GPU counterparts by relying on hardware-level optimizations, as already shown in this thesis and other studies [6, 7, 8, 188, 189].

In addition, modern SoC-FPGAs offer internal high-speed connections that allow CPUs, GPUs, and FPGAs to interact on the same die, thus reducing communication latency and power consumption. However, adoption has not yet occurred, primarily due to the complexity of hardware design and the lack of standards for interconnection, structure, and program description. In this respect, the HyperFPGA hardware and software provide a common reference by being modular, flexible, and open source, where different solutions can be implemented to appropriately measure their capabilities.

A survey of the most relevant heterogeneous clusters was carried out to determine the best design decisions for future implementations. As a result, the HyperFPGA was built, a unique scalable SoC-FPGA cluster that provides power monitoring tools along with FPGAs tightly coupled through diverse I/Os offering a high level of flexibility and introspection. Its programmable framework allows experimenting with diverse computing paradigms.

The platform was developed on the principles of openness, modularity, and scalability. Modularity was satisfied by developing an open-source hardware platform consisting of COTS SoMs and high-speed high-density connectors for scalability. The software environment consists of open-source custom tools that integrate existing applications for programming and management. Furthermore, vendor interoperability can be possible with a fixed SoM form factor and signal routing. A custom Linux OS was implemented to meet remote management requirements using *JupyterHub*. Using *JupyterHub*, management is greatly simplified and centralized to a single server. Additionally, parallel programming paradigms are implemented in the cluster using *JupyterLab* and the interconnection layers developed in-house consisting of the ComBlock, custom kernel driver, and XSA2Bit script.

The HyperFPGA software and hardware were tested by implementing a backtracking algorithm to solve the N-Queens problem. A speed-up of approximately 251 times with respect to a single ZedBoard was obtained without performing any major optimization, which shows the potential of the hardware and the efficiency of the development environment.

FPGA-based heterogeneous computing is a challenging field with enormous potential to change the dominant computing paradigm. In recent years, great interest has contributed to the development of tools and, more importantly, experimental platforms. With standard platform descriptions and interfaces, an open collaborative development approach will allow the creation of communities to accelerate adoption. New technologies such as SoC-FPGAs will certainly be at the center of future cluster architectures, considering the advantages of having CPUs, GPUs, and FPGAs on the same device.

6.1 Future works

Now that we have established that it is possible to program and interact with the HyperFPGA efficiently, it is important to proceed with more complex problems. Some of which are already in the works: a distributed multi-agent reinforced learning optimizer for quantum computing algorithms [190], and the study of massive asynchronous cellular automata machines [191, 192].

An aspect that is not discussed in this thesis is the FPGA-specific network. FPGAs are widely known for their low latency and high throughput, which in mixed networks may be lost due to the overhead of the communication protocols for CPUs and GPUs [125]. This part of the system offers significant improvements that will come from taking advantage of the seemingly scalable FPGA fabric that results from interconnecting the HyperFPGA nodes. To achieve this, high-speed and low-latency communication protocols will be used to take advantage of the high bandwidth and flexibility of the available I/Os. A generic distributed memory approach such as the one described in [179] could be assigned to the cluster to begin exploring direct communication between adjacent FPGAs and collaboration among all FPGAs in the HyperFPGA.

The HyperFPGA offers a variety of direct interconnections between nodes, MGTs, HP I/Os, and HD I/Os. MGTs provide the highest throughput at a low cost of link overhead, which was addressed with custom protocols in [1, 167]. Other important details are portability and openness of the interface, which is a special focus of Kyokko [167]. Although many multi-FPGA systems rely on standard protocols, custom protocols are the third most used and the first in terms of performance in the top500 supercomputers [193], in addition to the important contribution of custom protocols to scientific computing.

The HyperFPGA inter-FPGA network will have to support the following:

- An open and easily portable interface.
- Flexible topology to meet the transmission rate and data throughput required by the different problems and paradigms by using virtual channels as in VCSN [194] or fixed as in [77].
- As shown in Table 2.1, custom protocols remain relevant, suggesting that no industry standard fully satisfies the requirements of heterogeneous computing. A modular protocol reduces the effort required to optimize the interfaces, routers, and switches.

In addition to the concrete developments regarding the HyperFPGA FPGA-dedicated network, other open problems remain unaddressed. These can be taken from the SIRCA report [161] which divides the tools into four phases of development: formulation, design, translation, and execution.

The formulation phase consists of elaborating and optimizing algorithms for parallel computing at the highest level of abstraction, mostly dealing with pseudocodes and verbal language for reasoning. Formulation is the most critical step in which researchers can benefit the most from the knowledge of the chosen parallel paradigm. Tools that provide strategic exploration, high-level prediction, and numerical analysis have a strong positive impact on the other phases.

The design phase provides languages used to translate an algorithm into a behavioral implementation. This field has broadened with the creation of modern HDL and HLS languages, such as Chisel [155, 195] based on Scala and Clash [154, 196] based on Haskell, and high-level synthesis tools, such as BondManchine based on Go [197], and several HLS implementations [47, 198, 199, 198, 156]. New developments have solved, to some degree, the issues of portability and interoperability by raising abstraction. However, the method for scaling designs to heterogeneous clusters remains platform-specific, forcing users to take responsibility for porting and partitioning the designs. Furthermore, users are tasked with specifying a concurrency model at the system level. An in-depth study of design tools, frameworks, and strategies for design-space exploration can be found in [200].

Once a PC-compatible description of the algorithm is available, the next phase maps it onto the actual physical resources of the system. This phase is known as translation or place-and-route (PAR). Several improvements have occurred in recent years [201, 202]. Most studies focus on optimizing the performance of the process by implementing parallelism with good results compared to vendor tools. However, these improvements are not easily integrated into proprietary workflows and require a high level of expertise for effective usage. Likewise, existing PARs targeting clusters are platform-specific and will likely stay that way until a standard method of describing a heterogeneous system is adopted.

In the final phase, execution, developers must be able to verify and analyze the performance of the implementation. Critical runtime services must be included, such as task management, checkpoints, heartbeats, and debugging. The effective implementation of such services depends on their consideration in the previous design phases. The works studied in detail in [203] provided definitions of abstraction layers for user interaction and management, showing a great improvement in the execution phase. Similarly, several FPGA operating systems have been developed [204, 205, 206] implementing abstractions such as threads and processes.

These challenges can be highlighted in the following list:

- Abstract hardware model for different heterogeneous platforms and programming paradigms.
- Translation tools capable of targeting scalable heterogeneous platforms.

- Flexible design tools to optimize implementations targeting heterogeneous clusters.
- High-level prediction tools for performance, energy consumption, and resource utilization.
- Universal debugging and verification tools for distributed reconfigurable computing.

Even if there are platform-specific solutions to some of the aforementioned challenges, the real challenge is to develop standard and generic solutions suitable for any heterogeneous cluster implementation in a community-driven development approach that would greatly accelerate adoption and growth, as shown in [207, 208, 209].

Bibliography

[1] Roberto Sanchez Correa and Jean Pierre David. Ultra-low latency communication channels for fpga-based hpc cluster. *Integration*, 63:41–55, 2018. URL: `https://www.sciencedirect.com/science/article/pii/S0167926017303966`, `doi:https://doi.org/10.1016/j.vlsi.2018.05.005`.

[2] Tilak Agerwala. Exascale computing: The challenges and opportunities in the next decade. In *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pages 1–1, 2010. `doi:10.1109/HPCA.2010.5416662`.

[3] GAO. Advances Made Towards Implementing the National Strategy, but Better Reporting and a More Detailed Plan Are Needed. Technical report, GAO, 9 2021. URL: `https://www.gao.gov/assets/gao-21-104500.pdf`.

[4] EH D'Hollander. *Transition of HPC towards exascale computing*, volume 24. IOS Press, 2013.

[5] National Oceanic and Atmospheric Administration. High Performance Computing Strategic Plan 2015-2020. Technical report, National Oceanic and Atmospheric Administration, 9 2015. URL: `https://www.noaa.gov/sites/default/files/legacy/document/2020/Apr/HPCStrategy_Final_Draft_080913.pdf`.

[6] Murad Qasaimeh et al. Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–8, 2019. `doi:10.1109/ICESS.2019.8782524`.

[7] Kenneth O'Neal et al. Predictive modeling for CPU, GPU, and FPGA performance and power consumption: A survey. In *Proceedings of IEEE Computer Society Annual*

Symposium on VLSI, ISVLSI, volume 2018-July, pages 763–768. IEEE Computer Society, 8 2018. doi:10.1109/ISVLSI.2018.00143.

[8] Mike Southworth. Choosing the Best Processor for the Job. Technical report, Curtis-Wright, 10 2021. URL: https://www.curtisswrightds.com/sites/default/files/2021-10/Choosing-the-Best-Processor-for-the-Job-white-paper.pdf.

[9] Katherine Compton et al. Reconfigurable computing. *ACM Computing Surveys*, 34(2):171–210, 6 2002. URL: https://dl.acm.org/doi/10.1145/508352.508353, doi:10.1145/508352.508353.

[10] John D Davis. *FAST: A flexible architecture for simulation and testing of multiprocessor and CMP systems*. PhD thesis, Stanford University, 12 2006.

[11] Mario Porrmann et al. RAPTOR-A scalable platform for rapid prototyping and FPGA-based cluster computing. In *Advances in Parallel Computing*, volume 19. IOS Press, 2010. doi:10.3233/978-1-60750-530-3-592.

[12] Sandeep Kumar et al. Breaking ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4249 LNCS:101–118, 2006. doi:10.1007/11894063{_}9.

[13] Oskar Mencer et al. Cube: A 512-FPGA cluster. In *2009 5th Southern Conference on Programmable Logic (SPL)*, pages 51–57. IEEE, 4 2009. URL: http://ieeexplore.ieee.org/document/4914907/, doi:10.1109/SPL.2009.4914907.

[14] RIKEN Center for Computational Science. Fugaku: Riken's flagship supercomputer, 2020. URL: https://www.fugaku-riken.jp/.

[15] Kuen Hung Tsoi et al. Axel. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '10*, page 115, New York, New York, USA, 2010. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=1723112.1723134, doi:10.1145/1723112.1723134.

[16] Ra Inta et al. The "Chimera": An Off-The-Shelf CPU/GPGPU/FPGA Hybrid Computing Platform. *International Journal of Reconfigurable Computing*, 2012:1–10, 2012. URL: `http://www.hindawi.com/journals/ijrc/2012/241439/`, `doi:10.1155/2012/241439`.

[17] Krste Asanovic et al. The Landscape of Parallel Computing Research: A View from Berkeley. 2006. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html`.

[18] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. Google workloads for consumer devices: Mitigating data movement bottlenecks. *SIGPLAN Not.*, 53(2):316–331, mar 2018. URL: `https://doi-org.ezproxy.cern.ch/10.1145/3296957.3173177`, `doi:10.1145/3296957.3173177`.

[19] Kazi Asifuzzaman, Narasinga Rao Miniskar, Aaron R. Young, Frank Liu, and Jeffrey S. Vetter. A survey on processing-in-memory techniques: Advances and challenges. *Memories - Materials, Devices, Circuits and Systems*, 4:100022, 2023. URL: `https://www.sciencedirect.com/science/article/pii/S2773064622000160`, `doi:https://doi.org/10.1016/j.memori.2022.100022`.

[20] Charlie Demerjian. AMD outs Genoa HBM... Sort of... as a GPU, 7 2023. URL: `https://www.semiaccurate.com/2023/07/07/amd-outs-genoa-hbm-sort-of-as-a-gpu/`.

[21] Altera corporation. What is an SoC FPGA? Architecture Brief. Technical report, Altera, 7 2014. URL: `www.altera.com/socarchitecture`.

[22] Wim Vanderbauwhede et al. *High-Performance Computing Using FPGAs*. Springer, 2013.

[23] Mariette Awad. FPGA supercomputing platforms: A survey. In *2009 International Conference on Field Programmable Logic and Applications*, pages 564–568. IEEE, 8 2009. URL: `http://ieeexplore.ieee.org/document/5272406/`, `doi:10.1109/FPL.2009.5272406`.

[24] Clive Maxfield. Who made the first PLD? - EETimes, 9 2011. URL: `https://www.eetimes.com/who-made-the-first-pld/`.

[25] Clive Maxfield. Xilinx co-founder ross freeman honored - eetimes, 2017. URL: `https://www.eetimes.com/xilinx-co-founder-ross-freeman-honored/`.

[26] Xilinx Inc. *Vivado Design Suite User Guide*, 2021. Version 2021.1. URL: `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2021_1/ug973-vivado-release-notes-install-license.pdf`.

[27] Xilinx Inc. *Vitis Unified Software Platform User Guide*, 2021. Version 2021.1. URL: `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2021_1/ug1416-vitis-unified-platform.pdf`.

[28] Intel Corporation. *Quartus Prime User Guide*, 2021. Version 21.1. URL: `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-qps.pdf`.

[29] Microchip Technology Inc. *Libero SoC Design Suite User Guide*, 2021. Version 12.0. URL: `https://www.microsemi.com/document-portal/doc_view/131953-libero-soc-design-suite-v12-0-user-guide`.

[30] Yosys Development Team. Yosys open synthesis suite, 2021. Accessed on May 9, 2023. URL: `https://github.com/YosysHQ/yosys`.

[31] CHIPS Alliance. FOSS Flows For FPGA — F4PGA documentation, 2017. URL: `https://f4pga.readthedocs.io/en/latest/index.html`.

[32] Agile Analog Ltd. Rapidsilicon: Accelerating silicon development, 2021. Accessed on May 9, 2023. URL: `https://www.agileanalog.com/products/rapidsilicon`.

[33] Philip Colella. Defining Software Requirements for Scientific Computing, 2004. URL: `https://www.krellinst.org/doecsgf/conf/2013/pres/pcolella.pdf`.

[34] Roger D. Chamberlain. Architecturally truly diverse systems: A review. *Future Generation Computer Systems*, 110:33–44, 9 2020. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0167739X19313184`, `doi:10.1016/j.future.2020.03.061`.

[35] R. Palmer. Parallel Dwarfs (archived), 2011. URL: `https://web.archive.org/web/20210430053907/https://archive.codeplex.com/?p=paralleldwarfs`.

[36] Shuai Che et al. Rodinia: A benchmark suite for heterogeneous computing. *Proceedings of the 2009 IEEE International Symposium on Workload Characterization, IISWC 2009*, pages 44–54, 2009. `doi:10.1109/IISWC.2009.5306797`.

[37] Virginia Tech Synergy. GitHub - vtsynergy/OpenDwarfs: A benchmark suite., 2019. URL: `https://github.com/vtsynergy/OpenDwarfs`.

[38] Andrew Putnam et al. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In *ISCA '14: Proceeding of the 41st annual international symposium on Computer architecuture*, pages 13–24, 6 2014. `doi:10.5555/2665671.2665678`.

[39] Alibaba. Deep Dive into Alibaba Cloud F3 FPGA as a Service Instances - Alibaba Cloud Community, 2018. URL: `https://www.alibabacloud.com/blog/deep-dive-into-alibaba-cloud-f3-fpga-as-a-service-instances_594057`.

[40] Amazon. Amazon EC2 F1 Instances, 2017. URL: `https://aws.amazon.com/ec2/instance-types/f1/`.

[41] Rafał Kiełbik et al. ARUZ — Large-scale, massively parallel FPGA-based analyzer of real complex systems. *Computer Physics Communications*, 232:22–34, 11 2018. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0010465518302182`, `doi:10.1016/j.cpc.2018.06.010`.

[42] Farah Fahim et al. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. 3 2021. URL: `http://arxiv.org/abs/2103.05579`.

[43] Jason Villarreal, Adrian Park, Walid Najjar, and Robert Halstead. Designing modular hardware accelerators in c with roccc 2.0. In *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 127–134, 2010. `doi:10.1109/FCCM.2010.28`.

[44] Razvan Nane et al. Dwarv 2.0: A cosy-based c-to-vhdl hardware compiler. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 619–622, 2012. `doi:10.1109/FPL.2012.6339221`.

[45] Alexandros Papakonstantinou et al. Efficient compilation of cuda kernels for high-performance computing on fpgas. *ACM Trans. Embed. Comput. Syst.*, 13(2), 9 2013. URL: `https://doi-org.ezproxy.cern.ch/10.1145/2514641.2514652`, `doi:10.1145/2514641.2514652`.

[46] Andrew Canis et al. Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems. *ACM Trans. Embed. Comput. Syst.*, 13(2), 9 2013. URL: `https://doi-org.ezproxy.cern.ch/10.1145/2514740`, `doi:10.1145/2514740`.

[47] Seyong Lee et al. Openacc to fpga: A framework for directive-based high-performance reconfigurable computing. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 544–554, 2016. `doi:10.1109/IPDPS.2016.28`.

[48] E. Waingold et al. Baring it all to software: Raw machines. *Computer*, 30(9):86–93, 9 1997. URL: `http://ieeexplore.ieee.org/document/612254/`, `doi:10.1109/2.612254`.

[49] Heiko Kalte et al. A Prototyping Platform for Dynamically Reconfigurable System on Chip Designs. In *Proceedings of the IEEE Workshop Heterogeneous reconfigurable Systems on Chip (SoC)*, Hamburg, 2002.

[50] Craig Steffen et al. Nallatech In-Socket FPGA Front-Side Bus Accelerator. *Computing in Science & Engineering*, 12(02):78–83, 3 2010. `doi:10.1109/MCSE.2010.45`.

[51] Christopher Pohl et al. vMAGIC-Automatic Code Generation for VHDL. *International Journal of Reconfigurable Computing*, 9, 2009. URL: `http://vmagic.sourceforge.net/.`, `doi:10.1155/2009/205149`.

[52] Spyros Lyberis et al. FPGA prototyping of emerging manycore architectures for parallel programming research using Formic boards. *Journal of Systems Architecture*, 60(6):481–493, 6 2014. URL: `https://linkinghub.elsevier.com/retrieve/pii/S138376211400054X`, `doi:10.1016/j.sysarc.2014.03.002`.

[53] Spyros Lyberis et al. Formic: Cost-efficient and scalable prototyping of manycore architectures. In *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012*, pages 61–64, 2012. `doi:10.1109/FCCM.2012.20`.

[54] Shah Hemal et al. Remote Direct Memory Access (RDMA) Protocol Extensions. RFC 7306, RFC Editor, 6 2014. URL: `https://www.rfc-editor.org/rfc/rfc7306.txt`.

[55] Stamatis G. Kavadias et al. On-chip communication and synchronization mechanisms with cache-integrated network interfaces. *CF 2010 - Proceedings of the 2010 Computing Frontiers Conference*, pages 217–226, 2010. `doi:10.1145/1787275.1787328/FORMAT/PDF`.

[56] Vaibhav Kale. Using the MicroBlaze Processor to Accelerate Cost-Sensitive Embedded System Development. Technical report, Xilinx, 6 2016. URL: `https://docs.xilinx.com/v/u/en-US/wp469-microblaze-for-cost-sensitive-apps`.

[57] Cadence. Palladium Emulation | Cadence, 2019. URL: `https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html`.

[58] Siemens. Veloce Prototyping - FPGA | Siemens Software, 2022. URL: `https://eda.sw.siemens.com/en-US/ic/veloce/fpga-prototyping/`.

[59] Bruno da Silva et al. Comparing and combining GPU and FPGA accelerators in an image processing context. In *2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–4. IEEE, 9 2013. URL: `http://ieeexplore.ieee.org/document/6645552/`, `doi:10.1109/FPL.2013.6645552`.

[60] Takuya Otsuka et al. An Image Recognition System for Multiple Video Inputs over a Multi-FPGA System. In *2012 IEEE 6th International Symposium on Embedded Multicore SoCs*, pages 1–7. IEEE, 9 2012. URL: `http://ieeexplore.ieee.org/document/6354671/`, `doi:10.1109/MCSoC.2012.33`.

[61] The RTN Collaboration. 64-transputer machine. In *Procedings of CHEP*, pages 353–360, Geneva, 1992.

[62] H. Schmit et al. Behavioral synthesis for fpga-based computing. In *Proceedings of IEEE Workshop on FPGA's for Custom Computing Machines*, pages 125–132, 1994. `doi:10.1109/FPGA.1994.315591`.

[63] A. Cruz et al. SUE: A special purpose computer for spin glass models. *Computer Physics Communications*, 133(2-3):165–176, 1 2001. `doi:10.1016/S0010-4655(00)00170-3`.

[64] F. Belletti et al. Ianus: an adaptive FPGA computer. *Computing in Science & Engineering*, 8(1):41–49, 1 2006. URL: `http://ieeexplore.ieee.org/document/1563961/`, `doi:10.1109/MCSE.2006.9`.

[65] Francesco Belletti et al. Janus: An FPGA-Based System for High-Performance Scientific Computing. *Computing in Science & Engineering*, 11(1):48–58, 1 2009. URL: `https://ieeexplore.ieee.org/document/4720223/`, `doi:10.1109/MCSE.2009.11`.

[66] M. Baity-Jesi et al. An FPGA-based supercomputer for statistical physics: The weird case of Janus. In *High-Performance Computing Using FPGAs*, volume 9781461417910, pages 481–506. Springer New York, 3 2013. URL: `https://link-springer-com.ezproxy.cern.ch/chapter/10.1007/978-1-4614-1791-0_16`, `doi:10.1007/978-1-4614-1791-0{\_}16/TABLES/4`.

[67] Tim Güneysu et al. Cryptanalysis with COPACOBANA. *IEEE Transactions on Computers*, 57(11):1498–1513, 11 2008. URL: `http://ieeexplore.ieee.org/document/4515858/`, `doi:10.1109/TC.2008.80`.

[68] Wolfgang Kastl et al. A parallel computing system with specialized coprocessors for cryptanalytic algorithms. In Felix C. Freiling, editor, *P170 - Sicherheit 2010 - Sicherheit, Schutz und Zuverlässigkeit*, pages 78–83, Bonn, 2010. Gesellschaft für Informatik e.V. URL: `https://dl.gi.de/handle/20.500.12116/19801`.

[69] Bianca Danczul et al. Cuteforce Analyzer: A Distributed Bruteforce Attack on PDF Encryption with GPUs and FPGAs. In *2013 International Conference on Availability, Reliability and Security*, pages 720–725. IEEE, 9 2013. URL: `http://ieeexplore.ieee.org/document/6657310/`, `doi:10.1109/ARES.2013.94`.

[70] Anson H.T. Tse et al. Dynamic scheduling Monte-Carlo framework for multi-accelerator heterogeneous clusters. In *2010 International Conference on Field-Programmable Technology*, pages 233–240. IEEE, 12 2010. URL: `http://ieeexplore.ieee.org/document/5681495/`, `doi:10.1109/FPT.2010.5681495`.

[71] Guangming Tan et al. SuperDragon. *ACM Transactions on Reconfigurable Technology and Systems*, 8(4):1–22, 10 2015. URL: `https://dl.acm.org/doi/10.1145/2740966`, `doi:10.1145/2740966`.

[72] Simon W. Moore et al. Bluehive - A field-programable custom computing machine for extreme-scale real-time neural network simulation. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 133–140. IEEE, 4 2012. URL: `https://ieeexplore.ieee.org/document/6239804/`, `doi:10.1109/FCCM.2012.32`.

[73] Paul J Fox et al. Reliably prototyping large SoCs using FPGA clusters. In *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 5 2014. URL: `http://ieeexplore.ieee.org/document/6861350/`, `doi:10.1109/ReCoSoC.2014.6861350`.

[74] A. Theodore Markettos et al. Interconnect for commodity FPGA clusters: Standardized or customized? In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 9 2014. URL: `http://ieeexplore.ieee.org/document/6927472/`, `doi:10.1109/FPL.2014.6927472`.

[75] Rishiyur S Nikhil et al. *BSV by Example.* 10 edition, 2010. URL: `http://www.bluespec.com/support/`.

[76] M. Baity-Jesi et al. Janus II: A new generation application-driven computer for spin-system simulations. *Computer Physics Communications*, 185(2):550–559, 2 2014. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0010465513003470`, `doi:10.1016/j.cpc.2013.10.019`.

[77] Rafał Kiełbik et al. Methodology of Firmware Development for ARUZ—An FPGA-Based HPC System. 8 2020. URL: `www.mdpi.com/journal/electronics`, `doi:t10.3390/electronics9091482`.

[78] VHDL Preprocessor Home Page, 2006. URL: `https://sourceforge.net/projects/vhdlpp/`.

[79] Shvan Karim et al. AstroByte: Multi-FPGA Architecture for Accelerated Simulations of Spiking Astrocyte Neural Networks. *Proceedings of the 2020 Design, Automation and Test in Europe Conference and Exhibition, DATE 2020*, pages 1568–1573, 3 2020. `doi:10.23919/DATE48585.2020.9116312`.

[80] Shuangming Yang et al. BiCoSS: Toward Large-Scale Cognition Brain With Multigranular Neuromorphic Architecture. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2801–2815, 7 2022. `doi:10.1109/TNNLS.2020.3045492`.

[81] Damien Gratadour. Microgate - Green Flash, 2021. URL: `http://green-flash.lesia.obspm.fr/microgate.html`.

[82] Yann Clénet et al. MICADO-MAORY SCAO Preliminary design, development plan & calibration strategies. 2019. URL: `https://hal.archives-ouvertes.fr/hal-03078430`.

[83] Andrew Brown et al. Distributed Event-Based Computing. *Advances in Parallel Computing*, 32:583–592, 2018. URL: `https://ebooks.iospress.nl/doi/10.3233/978-1-61499-843-3-583`, `doi:10.3233/978-1-61499-843-3-583`.

[84] Mihai A. Petrovici et al. Characterization and Compensation of Network-Level Anomalies in Mixed-Signal Neuromorphic Modeling Platforms. *PLOS ONE*, 9(10):e108590, 10 2014. URL: `https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0108590`, `doi:10.1371/JOURNAL.PONE.0108590`.

[85] Itta Ohmura et al. MDGRAPE-4: a special-purpose computer system for molecular dynamics simulations. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 372(2021), 8 2014. URL: `/pmc/articles/PMC4084528//pmc/articles/PMC4084528/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC4084528/`, `doi:10.1098/RSTA.2013.0387`.

[86] Jagath Weerasinghe et al. Enabling FPGAs in Hyperscale Data Centers. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 1078–1086. IEEE, 8 2015. URL: `https://ieeexplore.ieee.org/document/7518378/`, `doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.199`.

[87] Xilinx. Xilinx and IBM to Enable FPGA-Based Acceleration within SuperVessel OpenPOWER Development Cloud, 20016. URL: `https://www.xilinx.com/news/press/2016/xilinx-and-ibm-to-enable-fpga-based-acceleration-within-supervessel-openpower-development-cloud.html`.

[88] Francois Abel et al. An FPGA Platform for Hyperscalers. In *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*, pages 29–32. IEEE, 8 2017. URL: `http://ieeexplore.ieee.org/document/8071053/`, `doi:10.1109/HOTI.2017.13`.

[89] Jagath Weerasinghe et al. Disaggregated FPGAs: network performance comparison against bare-metal servers, virtual machines and linux containers. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, 0:9–17, 7 2016. `doi:10.1109/CLOUDCOM.2016.0018`.

[90] Burkhard Ringlein, Francois Abel, Alexander Ditter, Beat Weiss, Christoph Hagleitner, and Dietmar Fey. Programming reconfigurable heterogeneous computing clusters using mpi with transpilation. In *2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, pages 1–9, 2020. `doi:10.1109/H2RC51942.2020.00006`.

[91] Burkhard Ringlein, Francois Abel, Alexander Ditter, Beat Weiss, Christoph Hagleitner, and Dietmar Fey. Zrlmpi: A unified programming model for reconfigurable heterogeneous computing clusters. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 220–220, 2020. `doi:10.1109/FCCM48280.2020.00051`.

[92] Juan Miguel de Haro, Jaume Bosch, Antonio Filgueras, Miquel Vidal, Daniel Jiménez-González, Carlos Álvarez, Xavier Martorell, Eduard Ayguadé, and Jesús Labarta. Ompss@fpga framework for high performance fpga computing. *IEEE Transactions on Computers*, 70(12):2029–2042, 2021. `doi:10.1109/TC.2021.3086106`.

[93] Juan Miguel de Haro, Rubén Cano, Carlos Álvarez, Daniel Jiménez-González, Xavier Martorell, Eduard Ayguadé, Jesús Labarta, Francois Abel, Burkhard Ringlein, and Beat Weiss. Ompss@cloudfpga: An fpga task-based programming model with message passing. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 828–838, 2022. `doi:10.1109/IPDPS53621.2022.00085`.

[94] Hafsah Shahzad et al. Survey and future trends for fpga cloud architectures. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–10, 2021. `doi:10.1109/HPEC49654.2021.9622807`.

[95] Christophe Bobda et al. The future of fpga acceleration in datacenters and the cloud. *ACM Trans. Reconfigurable Technol. Syst.*, 15(3), 2 2022. doi:10.1145/3506713.

[96] Ron Sass et al. Reconfigurable Computing Cluster (RCC) project: Investigating the feasibility of FPGA-based petascale computing. In *Proceedings 2007 IEEE Symposium on Field-Programme Custom Computing Machines, FCCM 2007*, 2007. URL: https://ieeexplore.ieee.org/document/4297250, doi:10.1109/FCCM.2007.62.

[97] Andrew G. Schmidt et al. Reconfigurable computing cluster project: Phase I brief. *Proceedings of the 16th IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM'08*, pages 300–301, 2008. doi:10.1109/FCCM.2008.12.

[98] AMD Xilinx. Aurora 64B/66B LogiCORE IP Product Guide, 10 2022. URL: https://docs.xilinx.com/r/en-US/pg074-aurora-64b66b.

[99] Ranjesh G. Jaganathan et al. A configurable network protocol for cluster based communications using modular hardware primitives on an intelligent NIC. *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, SC 2003*, 2003. doi:10.1145/1048935.1050173.

[100] HPC Open. Open MPI: Open Source High Performance Computing, 2022. URL: https://www.open-mpi.org/.

[101] Kushal Datta et al. RBoot: Software infrastructure for a remote FPGA Laboratory. *Proceedings 2007 IEEE Symposium on Field-Programme Custom Computing Machines, FCCM 2007*, pages 343–344, 2007. doi:10.1109/FCCM.2007.53.

[102] Staff. FPGA High-Performance Computing Alliance (FHPCA) (archived), 7 2005. URL: https://web.archive.org/web/20060622011428/http://www.fhpca.org/index.html.

[103] R. Baxter et al. Maxwell - a 64 FPGA Supercomputer. In *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pages 287–294. IEEE, 8 2007. URL: http://ieeexplore.ieee.org/document/4291933/, doi:10.1109/AHS.2007.71.

[104] Rob Baxter et al. The FPGA high-performance computing alliance parallel toolkit. *Proceedings - 2007 NASA/ESA Conference on Adaptive Hardware and Systems, AHS-2007*, pages 301–307, 2007. doi:10.1109/AHS.2007.104.

[105] Michael Showerman et al. QP: A Heterogeneous Multi-Accelerator Cluster. In *10th LCI International Conference on High-Performance Clustered Computing*, Boulder, Colorado, 3 2009.

[106] Xilinx. ISE Design suite, 2013. URL: `https://www.xilinx.com/products/design-tools/ise-design-suite.html`.

[107] Avneesh Pant et al. Phoenix: A runtime environment for high performance computing on chip multiprocessors. In *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2009*, pages 119–126, 2009. `doi:10.1109/PDP.2009.41`.

[108] Adaptive Computing Enterprises Inc. TORQUE Resource Manager Administrator Guide 4.2.10. Technical report, 2015. URL: `www.adaptivecomputing.com`.

[109] Adaptive Computing Enterprises Inc. Maui Scheduler™ Administrator's Guide, 2014. URL: `http://docs.adaptivecomputing.com/maui/`.

[110] Alan George et al. Novo-G: At the Forefront of Scalable Reconfigurable Supercomputing. *Computing in Science & Engineering*, 13(1):82–86, 1 2011. URL: `http://ieeexplore.ieee.org/document/5678570/`, `doi:10.1109/MCSE.2011.11`.

[111] Alan D. George et al. Novo-G#: Large-scale reconfigurable computing with direct and programmable interconnects. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 9 2016. URL: `http://ieeexplore.ieee.org/document/7761639/`, `doi:10.1109/HPEC.2016.7761639`.

[112] Xilinx. Interlaken 150G. Technical report, Xilinx, 10 2017. URL: `https://docs.xilinx.com/v/u/en-US/pg212-interlaken-150g`.

[113] Roberto Giorgi. AXIOM: A 64-bit reconfigurable hardware/software platform for scalable embedded computing. In *2017 6th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4. IEEE, 6 2017. URL: `http://ieeexplore.ieee.org/document/7977173/`, `doi:10.1109/MECO.2017.7977173`.

[114] Carlos Álvarez et al. The AXIOM software layers. *Microprocessors and Microsystems*, 47:262–277, 11 2016. `doi:10.1016/J.MICPRO.2016.07.002`.

[115] Roberto Giorgi et al. AXIOM: A Scalable, Efficient and Reconfigurable Embedded Platform. In *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019*. IEEE, 2019. `doi:10.23919/DATE.2019.8715168`.

[116] Antonio Filgueras et al. The AXIOM project: IoT on heterogeneous embedded platforms. *IEEE Design and Test*, 38(5):74–81, 10 2021. `doi:10.1109/MDAT.2019.2952335`.

[117] AMD-Xilinx. Xilinx Adaptive Compute Clusters (XACC) Academia-Industry Research Ecosystem | HACC Resources, 2021. URL: `https://www.amd-haccs.io/adapt_2021.html`.

[118] AMD-Xilinx. Heterogeneous Accelerated Compute Clusters | HACC Resources, 2016. URL: `https://www.amd-haccs.io/index.html`.

[119] Timothy Prickett. Forging A Hybrid CPU-FPGA Supercomputer, 2018. URL: `https://www.nextplatform.com/2018/09/25/forging-a-hybrid-cpu-fpga-supercomputer/`.

[120] Paderborn Center for Parallel Computing (PC2). PC2 - Noctua 2 (Universität Paderborn), 2022. URL: `https://pc2.uni-paderborn.de/hpc-services/available-systems/noctua2`.

[121] Intel. oneAPI: A New Era of Heterogeneous Computing, 2022. URL: `https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html`.

[122] David Cock et al. Enzian: An Open, General, CPU/FPGA Platform for Systems Software Research. In *ASPLOS 2022: Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, page 18. ACM, 2 2022. `doi:10.1145/3503222.3507742`.

[123] Aggelos D. Ioannou et al. Unilogic: A novel architecture for highly parallel reconfigurable systems. *ACM Trans. Reconfigurable Technol. Syst.*, 13(4), 9 2020. `doi:10.1145/3409115`.

[124] Cygnus Consortium. About cygnus, 2018. URL: `https://www.ccs.tsukuba.ac.jp/wp-content/uploads/sites/14/2018/12/About-Cygnus.pdf`.

[125] Taisuke Boku et al. Cygnus - world first multihybrid accelerated cluster with gpu and fpga coupling. ICPP Workshops '22, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3547276.3548629`.

[126] Kohei Kikuchi et al. Implementation and performance evaluation of collective communications using circus on multiple fpgas. In *Proceedings of the HPC Asia 2023 Workshops*, HPC Asia '23 Workshops, page 15–23, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3581576.3581602`.

[127] Kentaro Sano et al. Essper: Elastic and scalable fpga-cluster system for high-performance reconfigurable computing with supercomputer fugaku. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia '23, page 140–150, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3578178.3579341`.

[128] John Davis et al. BEE3: Revitalizing Computer Architecture Research. Technical report, Microsoft, 4 2009. URL: `https://www.microsoft.com/en-us/research/publication/bee3-revitalizing-computer-architecture-research/`.

[129] Kimmo Kuusilinna et al. Designing BEE: A Hardware Emulation Engine for Signal Processing in Low-Power Wireless Applications. *EURASIP Journal on Applied Signal Processing*, 6:502–513, 2003. `doi:10.1155/s1110865703212154`.

[130] S.C. Jain et al. Evaluation of various routing architectures for multi-FPGA boards. In *VLSI Design 2000. Wireless and Digital Imaging in the Millennium. Proceedings of 13th International Conference on VLSI Design*, pages 262–267. IEEE Comput. Soc, 2000. URL: `http://ieeexplore.ieee.org/document/812619/`, `doi:10.1109/ICVD.2000.812619`.

[131] Chen Chang et al. Implementation of BEE: a real-time large-scale hardware emulation engine. *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays - FPGA '03*, pages 91–99, 2 2003. `doi:10.1145/611817.611832`.

[132] Chen Chang et al. BEE2: A high-end reconfigurable computing system. *IEEE Design and Test of Computers*, 22(2):114–125, 3 2005. `doi:10.1109/MDT.2005.30`.

[133] Andrew G. Schmidt et al. Checkpoint/restart and beyond: Resilient high performance computing with FPGAs. *Proceedings - IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2011*, pages 162–169, 2011. `doi:10.1109/FCCM.2011.22`.

[134] Scott Buscemi et al. Design and utilization of an FPGA cluster to implement a digital wireless channel emulator. *Proceedings - 22nd International Conference on Field Programmable Logic and Applications, FPL 2012*, pages 635–638, 2012. doi:10.1109/FPL.2012.6339253.

[135] Scott Buscemi et al. Design of a scalable digital Wireless Channel Emulator for networking radios. In *2011 - MILCOM 2011 Military Communications Conference*, pages 1858–1863. IEEE, 11 2011. URL: http://ieeexplore.ieee.org/document/6127583/, doi:10.1109/MILCOM.2011.6127583.

[136] David A. Patterson. RAMP: Research accelerator for multiple processors - A community vision for a shared experimental parallel HW/SW platform. In *ISPASS 2006: IEEE International Symposium on Performance Analysis of Systems and Software, 2006*, volume 2006, page 1, 2006. doi:10.1109/ISPASS.2006.1620784.

[137] Wirbel Loring. Berkeley Emulation Engine update - EDN, 5 2010. URL: https://www.edn.com/berkeley-emulation-engine-update/.

[138] Joseph Rothman et al. BEE technology overview. In *2012 International Conference on Embedded Computer Systems (SAMOS)*, pages 277–277, Samos, 1 2013. Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/SAMOS.2012.6404186.

[139] EDN. DESIGN TOOLS - BEEcube launches BEE4, a full-speed FPGA prototyping platform - EDN, 6 2010. URL: https://www.edn.com/design-tools-beecube-launches-bee4-a-full-speed-fpga-prototyping-platform/.

[140] Mingjie Lin. Hardware-Assisted Large-Scale Neuroevolution for Multiagent Learning. Technical report, University of Central Florida, Orlando, Florida, 12 2014. URL: https://apps.dtic.mil/sti/citations/ADA621804.

[141] Sokol, Iliza. NI's BEEcube Acquisition Drives 5G Communications | Microwaves & RF, 4 2015. URL: https://www.mwrf.com/technologies/systems/article/21846169/nis-beecube-acquisition-drives-5g-communications.

[142] National Instruments. What Is FlexRIO? - NI, 2022. URL: https://www.ni.com/it-it/shop/electronic-test-instrumentation/flexrio/what-is-flexrio.html.

[143] Leonardo Bonati et al. Colosseum: Large-Scale Wireless Experimentation Through Hardware-in-The-Loop Network Emulation. *2021 IEEE International Symposium on Dynamic Spectrum Access Networks, DySPAN 2021*, pages 105–113, 2021. doi:10.1109/DYSPAN53946.2021.9677430.

[144] Ettus. USRP Hardware Driver and USRP Manual: USRP X3x0 Series, 2014. URL: https://files.ettus.com/manual/page_usrp_x3x0.html.

[145] NI. ATCA Overview - NI, 2022. URL: https://www.ni.com/docs/en-US/bundle/atca-3671-getting-started/page/overview.html.

[146] Jung Ho Ahn et al. HyperX: topology, routing, and packaging of efficient large-scale networks. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09*, page 1, New York, New York, USA, 2009. ACM Press. URL: http://dl.acm.org/citation.cfm?doid=1654059.1654101, doi:10.1145/1654059.1654101.

[147] Sugandha Gupta et al. Getting Started with RFNoC in UHD 4.0 - Ettus Knowledge Base, 2022. URL: https://kb.ettus.com/Getting_Started_with_RFNoC_in_UHD_4.0.

[148] Ashish Chaudhari et al. A Scalable FPGA Architecture for Flexible, Large-Scale, Real-Time RF Channel Emulation. In *2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 7 2018. URL: https://ieeexplore.ieee.org/document/8449390/, doi:10.1109/ReCoSoC.2018.8449390.

[149] J. J. Dongarra et al. High-performance computing systems: Status and outlook. *Acta Numerica*, 21:379–474, 5 2012. doi:10.1017/S0962492912000050.

[150] Lamees M. Al Qassem et al. Fpgaaas: A survey of infrastructures and systems. *IEEE Transactions on Services Computing*, 15(2):1143–1156, 2022. doi:10.1109/TSC.2020.2976012.

[151] Antoniette Mondigo et al. Comparison of Direct and Indirect Networks for High-Performance FPGA Clusters. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume

12083 LNCS, pages 314–329. Springer, 2020. URL: http://link.springer.com/10.1007/978-3-030-44534-8_24, doi:10.1007/978-3-030-44534-8_24.

[152] Damien Gratadour, James Osborn, Hugues Deneux, Roberto Biasi, Denis Perret, Arnaud Sevin, Timothy J. Morris, Edward J. Younger, Christophe Rouaud, Jean-Tristan M. Buey, Damien Pretet, Jerome Lemaitre, Paolo Palazzari, Christian Patauner, Matthew J. Townson, Deli Geng, Lazar Staykov, Alastair G. Basden, Dietrich Pescoller, Mario Andrighettoni, Julien Bernard, Maxime Lainé, Florian Ferreira, and Nicolas Doucet. Prototyping AO RTC using emerging high performance computing technologies with the green flash project. In Dirk Schmidt, Laura Schreiber, and Laird M. Close, editors, *Adaptive Optics Systems VI*. SPIE, July 2018. doi:10.1117/12.2312686.

[153] A George et al. Novo-G: A View at the HPC Crossroads for Scientific Computing. In *ERSA Keynote for Reconfigurable Supercomputing Panel*, pages 21–30, 2010. URL: http://plaza.ufl.edu/poppyc/ERS5029.pdf.

[154] C. Baaij. *CλasH : from Haskell to hardware.* PhD thesis, University of Twente, Twente, 12 2009. URL: http://essay.utwente.nl/59482/.

[155] Jonathan Bachrach et al. Chisel: Constructing hardware in a Scala embedded language. In *Proceedings - Design Automation Conference*, pages 1216–1225, 2012. doi:10.1145/2228360.2228584.

[156] Grigory Chirkov and David Wentzlaff. Smappic: Scalable multi-fpga architecture prototype platform in the cloud. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 733–746, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3575693.3575753.

[157] Jovica Đurković, Vuk Vuković, and Lazar Raković. Open source approach in software development-advantages and disadvantages. URL: https://api.semanticscholar.org/CorpusID:14570097.

[158] Andres Cicuttin et al. HyperFPGA: A possible general purpose reconfigurable hardware for custom supercomputing. *2016 International Conference on Advances in Electrical,*

Electronic and Systems Engineering, ICAEES 2016, pages 21–26, 3 2017. doi:10.1109/ICAEES.2016.7888002.

[159] John W. Lockwood et al. NetFPGA - An open platform for gigabit-rate network switching and routing. In *Proceedings - MSE 2007: 2007 IEEE International Conference on Microelectronic Systems Education: Educating Systems Designers for the Global Economy and a Secure World*, pages 160–161, 2007. doi:10.1109/MSE.2007.69.

[160] Noa Zilberman et al. NetFPGA SUME: Toward 100 Gbps as research commodity. *IEEE Micro*, 34(5):32–41, 9 2014. doi:10.1109/MM.2014.61.

[161] Tarek El-Ghazawi et al. Exploration of a Research Roadmap for Application Development and Execution on Field-Programmable Gate Array (FPGA)-Based Systems. Technical Report ADA494473, George Washington University, Washington DC, USA, 10 2008. URL: https://apps.dtic.mil/sti/citations/ADA494473.

[162] AMD Versal Chip, 2023. URL: https://www.xilinx.com/products/silicon-devices/acap/versal.html.

[163] Susanne Kunath. Trenz te0803 trm, 2023. (Accessed on: 02/10/2023). URL: https://wiki.trenz-electronic.de/display/PD/TE0803+TRM.

[164] Sundance. Som3 - mpf300t, 2023. (Accessed on: 02/19/2023). URL: https://www.sundancedsp.com/som/polarfire/som-pf2-system-on-module-based-on-polarfire-mpf300t-1fcg784e/.

[165] Microchip Inc. *PolarFire SoC MSS Technical Reference Manual*, 2023. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/ReferenceManuals/PolarFire_SoC_FPGA_MSS_Technical_Reference_Manual_VC.pdf.

[166] Texas Instruments. *INA226 Bi-Directional Current/Power Monitor with I2C Interface*, 2015. URL: https://www.ti.com/product/INA226.

[167] Akinobu Tomori and Yasunori Osana. Kyokko: A vendor-independent high-speed serial communication controller. In *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, HEART '21, New York, NY, USA,

2021. Association for Computing Machinery. URL: `https://doi-org.ezproxy.cern.ch/10.1145/3468044.3468051`, `doi:10.1145/3468044.3468051`.

[168] Nekija Dzemaili. A reliable booting system for Zynq Ultrascale+ MPSoC devices. 2021. Presented 17 Mar 2021. URL: `https://cds.cern.ch/record/2763095`.

[169] Marvin Fuchs, Luis E. Ardila-Perez, Torben Mehner, and Oliver Sander. Split boot – true network-based booting on heterogeneous mpsocs. 2023. `arXiv:arXiv:2301.05642`, `doi:10.1109/TNS.2023.3237968`.

[170] Kasun S. Mannatunga, Luis G.G. Ordóñez, Marie B. Amador, Maria Liz Crespo, Andres Cicuttin, Stefano Levorato, Rodrigo Melo, and Bruno Valinoti. Design for portability of reconfigurable virtual instrumentation. In *2019 X Southern Conference on Programmable Logic (SPL)*, pages 45–52, 2019. `doi:10.1109/SPL.2019.8714446`.

[171] NERSC. sshspawner, 9 2016. URL: `https://github.com/NERSC/sshspawner`.

[172] Maria Liz Crespo, François Foulon, Andres Cicuttin, Mladen Bogovac, Clement Onime, Cristian Sisterna, Rodrigo Melo, Werner Florian Samayoa, Luis Guillermo García Ordóñez, Romina Molina, and Bruno Valinoti. Remote laboratory for e-learning of systems on chip and their applications to nuclear and scientific instrumentation. *Electronics*, 10(18), 2021. URL: `https://www.mdpi.com/2079-9292/10/18/2191`, `doi:10.3390/electronics10182191`.

[173] Maria Liz Crespo, A. Cicuttin, J.D.D. Gazzano, and F.R. Calle. Reconfigurable virtual instrumentation based on FPGA for science and high-education. In Jan Fagerberg, David C. Mowery, and Richard R. Nelson, editors, *Field-Programmable Gate Array (FPGA) Technologies for High Performance Instrumentation*, chapter 5, pages 99–123. IGI Global, 2016.

[174] ICTP-Mlab and INTI-CMNT. The core comblock, 2021. URL: `https://gitlab.com/rodrigomelo9/core-comblock`.

[175] Project Jupyter, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, Kyle Kelley, Gladys Nalvarte, Andrew Osheroff, M Pacer, Yuvi Panda, Fernando Perez, Benjamin Ragan Kelley, and Carol Willing. Binder 2.0

- Reproducible, interactive, sharable environments for science at scale. In Fatih Akici, David Lippa, Dillon Niederhut, and M Pacer, editors, *Proceedings of the 17th Python in Science Conference*, pages 113 – 120, 2018. doi:10.25080/Majora-4af1f417-011.

[176] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007. URL: https://ipython.org, doi:10.1109/MCSE.2007.53.

[177] Lisandro Dalcin and Yao-Lung L. Fang. mpi4py: Status update after 12 years of development. *Computing in Science & Engineering*, 23(4):47–54, 2021. doi:10.1109/MCSE.2021.3083216.

[178] Luis Guillermo García, Maria Liz Crespo, Sergio Carrato, Andres Cicuttin, Werner Florian, Romina Molina, Bruno Valinoti, and Stefano Levorato. High voltage isolated bidirectional network interface for soc-fpga based devices. In Sergio Saponara and Alessandro De Gloria, editors, *Applications in Electronics Pervading Industry, Environment and Society*, pages 280–285, Cham, 2021. Springer International Publishing.

[179] Werner Florian, Bruno Valinoti, Luis G. García, Marcos Cervetto, Edgardo Marchi, Maria Liz Crespo, Sergio Carrato, and Andres Cicuttin. An open-source hardware/software architecture for remote control of SoC-FPGA based systems. In *Lecture Notes in Electrical Engineering*, pages 69–75. Springer International Publishing, New York, USA, 2022. URL: https://doi.org/10.1007%2F978-3-030-95498-7_10, doi:10.1007/978-3-030-95498-7_10.

[180] Jamie K Infantolino and Mikayla Malley. Integrating the nqueens algorithm into a parameterized benchmark suite, 2016. URL: https://apps.dtic.mil/sti/pdfs/AD1003173.pdf.

[181] W. W. Rouse Ball. *Mathematical Recreations and Essays.* 1892. URL: https://web.northeastern.edu/seigen/11Magic/Books/Rouse%20Ball.pdf.

[182] OEIS Foundation Inc. Number of ways of placing n nonattacking queens on an n X n board., 2023. URL: https://oeis.org/A000170.

[183] Jordan Bell and Brett Stevens. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309(1):1–31, January 2009. doi:10.1016/j.disc.2007.12.043.

[184] Matteo Fischetti and Domenico Salvagnin. Finding first and most-beautiful queens by integer programming, 2019. `arXiv:1907.08246`.

[185] O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors. *Structured Programming*. Academic Press Ltd., GBR, 1972.

[186] Marcin Łajtar. The N queens problem - new variants of the Wirth algorithm. *Annales UMCS, Informatica*, 13(1), January 2013. `doi:10.2478/v10065-012-0013-3`.

[187] AVNET. ZedBoard. URL: `https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/`.

[188] Julio Daniel Dondo Gazzano et al. *Field-programmable gate array (FPGA) technologies for high performance instrumentation*. IGI Global, 7 2016. `doi:10.4018/978-1-5225-0299-9`.

[189] Julio Proaño Orellana et al. Diseño de una arquitectura heterogénea para la gestión eficiente de recursos FPGA en un cloud privado. In *Aplicaciones e innovación de la ingeniería en ciencia y tecnología*, pages 165–199. Editorial Abya-Yala, 2019. `doi:10.7476/9789978104910.0007`.

[190] Agustin Silva, Omar Gustavo Zabaleta, and Constancio Miguel Arizmendi. Learning mixed strategies in quantum games with imperfect information. *Quantum Reports*, 4(4):462–475, 2022. URL: `https://www.mdpi.com/2624-960X/4/4/33`, `doi:10.3390/quantum4040033`.

[191] A. Cicuttin, L. De Micco, M. L. Crespo, M. Antonelli, L. Garcia, W. Florian Samayoa, and A. Silva. Looking for suitable rules for true random number generation with asynchronous cellular automata. *Nonlinear Dynamics*, 111(3):2711–2722, October 2022. `doi:10.1007/s11071-022-07957-8`.

[192] A. Cicuttin, L. De Micco, M. L. Crespo, M. Antonelli, L. Garcia, and W. Florian. Physical implementation of asynchronous cellular automata networks: mathematical models and preliminary experimental results. *Nonlinear Dynamics*, 105(3):2431–2452, July 2021. `doi:10.1007/s11071-021-06754-z`.

[193] Top500 supercomputer site, 2016. URL: `https://www.top500.org/lists/top500/2023/06/`.

[194] Tomohiro Ueno and Kentaro Sano. Vcsn: Virtual circuit-switching network for flexible and simple-to-operate communication in hpc fpga cluster. 16(2), 3 2023. `doi:10.1145/3579848`.

[195] Adam Izraelevitz et al. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, volume 2017-November, pages 209–216. Institute of Electrical and Electronics Engineers Inc., 12 2017. `doi:10.1109/ICCAD.2017.8203780`.

[196] M. Kooijman. *Haskell as a higher order structural hardware description language*. PhD thesis, University of Twente, Twente, 12 2009. URL: `http://essay.utwente.nl/59381/`.

[197] Mirko Mariotti et al. The BondMachine, a moldable computer architecture. *Parallel Computing*, 109:102873, 3 2022. `doi:10.1016/J.PARCO.2021.102873`.

[198] Yasunori Osana et al. Openfc: a portable toolkit for custom fpga accelerators and clusters. In *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, pages 185–190, 2020. `doi:10.1109/CANDARW51189.2020.00045`.

[199] Roger D. Chamberlain et al. Auto-pipe: Streaming applications on architecturally diverse systems. *Computer*, 43(3):42–49, 2010. `doi:10.1109/MC.2010.62`.

[200] Romina Soledad Molina et al. High-Level Synthesis Hardware Design for FPGA-Based Accelerators: Models, Methodologies, and Frameworks. *IEEE Access*, 10:90429–90455, 2022. URL: `https://ieeexplore.ieee.org/document/9864576/`, `doi:10.1109/ACCESS.2022.3201107`.

[201] Yun Zhou et al. Accelerating FPGA Routing through Algorithmic Enhancements and Connection-aware Parallelization. In *ACM Transactions on Reconfigurable Technology and Systems*, volume 13. ACM PUB27 New York, NY, USA, 8 2020. URL: `https://dl.acm.org/doi/10.1145/3406959`, `doi:10.1145/3406959`.

[202] Mariya A. Zapletina et al. The Acceleration Techniques for the Modified Pathfinder Routing Algorithm on an Island-Style FPGA. In *2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, pages 920–923. IEEE, 1 2022. URL: `https://ieeexplore.ieee.org/document/9755536/`, `doi:10.1109/ElConRus54750.2022.9755536`.

[203] Anuj Vaishnav et al. A survey on FPGA virtualization. In *Proceedings - 2018 International Conference on Field-Programmable Logic and Applications, FPL 2018*, pages 131–138. Institute of Electrical and Electronics Engineers Inc., 11 2018. `doi:10.1109/FPL.2018.00031`.

[204] Kermin Fleming et al. The leap fpga operating system. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2014. `doi:10.1109/FPL.2014.6927488`.

[205] Lennart Clausing and Marco Platzner. Reconos64: A hardware operating system for modern platform fpgas with 64-bit support. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 120–127, 2022. `doi:10.1109/IPDPSW55747.2022.00029`.

[206] Dario Korolija et al. Do OS abstractions make sense on FPGAs? Do OS abstractions make sense on FPGAs? In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation*, pages 991–1010, 2020. URL: `https://www.usenix.org/conference/osdi20/presentation/roscoe`, `doi:10.5555/3488766.3488822`.

[207] Steffen Möller et al. Community-driven development for computational biology at Sprints, Hackathons and Codefests. *BMC bioinformatics*, 15 Suppl 14(14):S7, 11 2014. URL: `http://www.ncbi.nlm.nih.gov/pubmed/25472764http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4255748`, `doi:10.1186/1471-2105-15-S14-S7`.

[208] Mohashin Pathan et al. A novel community driven software for functional enrichment analysis of extracellular vesicles data. *Journal of Extracellular Vesicles*, 6(1):1321455, 12 2017. URL: `https://onlinelibrary.wiley.com/doi/10.1080/20013078.2017.1321455`, `doi:10.1080/20013078.2017.1321455`.

[209] Markus Kühbach et al. Community-Driven Methods for Open and Reproducible Software Tools for Analyzing Datasets from Atom Probe Microscopy. *Microscopy and Microanalysis*, 28(4):1038–1053, 8 2022. URL: `https://www.cambridge.org/core/product/identifier/S1431927621012241/type/journal_article`, doi: `10.1017/S1431927621012241`.