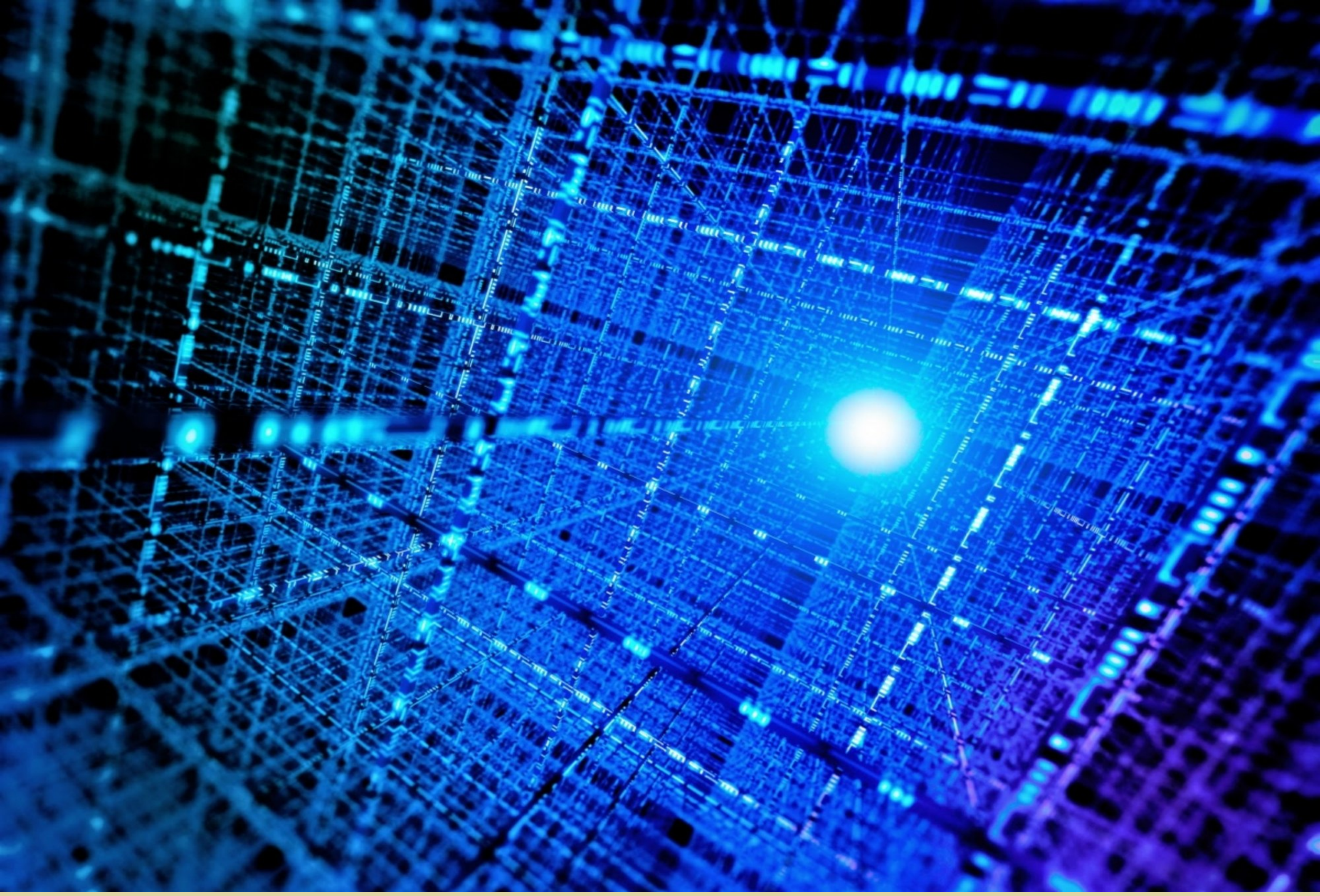


José Carlos Baptista Pereira Mendes Direito

Probabilistic Computing Using OpenCL on an FPGA Mini-cluster

Dissertation submitted to the Electrical and Computer Engineering Department of the Faculty of Science and Technology of the University of Coimbra in partial fulfillment of the requirements for the Degree of Master of Computer Science.

May 2018

· U C ·

Universidade de Coimbra

Probabilistic Computing Using OpenCL on an FPGA Mini-cluster

FCTUC

José Carlos Baptista Pereira Mendes Direito

Department of Electrical and Computer Engineering
Faculty of Sciences and Technology
University of Coimbra

Supervisor: Prof. Doctor Jorge Nuno de Almeida e Sousa Almada Lobo

Jury:
Prof. Doctor Vítor Manuel Mendes da Silva
Prof. Doctor Fernando Manuel dos Santos Perdigão
Prof. Doctor Jorge Nuno de Almeida e Sousa Almada Lobo

Dissertation submitted to the Electrical and Computer Engineering Department of the Faculty of Science and Technology of the University of Coimbra in partial fulfillment of the requirements for the Degree of Master of Computer Science.

May 2018

Acknowledgements

First, I wish to thank my supervisor, Prof. Doctor Jorge Lobo, at the University of Coimbra, for his steady guidance throughout this work. Second, would like to thank Fernando, Gonçalo, Bruno, Miguel and Hugo for their companionship during the long hours spent at the Mobile Robotics Laboratory. Finally, I would like to thank my family and my girlfriend for their never ending support and patience.

Obrigado,

Zé

Abstract

This thesis studies the implementation of Bayesian Inference on Re-configurable Hardware (FPGAs) using a Top Down approach. We started from a generic implementation targeting a CPU and, using the general purpose parallel programming language OpenCL, offloaded the computation bottlenecks to FPGAs installed on accelerator boards. A generic localization problem was implemented on an heterogeneous computing platform containing two Intel Xeon E5 CPUs and four Intel (formerly Altera) Stratix V FPGAs. The full capabilities of such a platform were extracted by the careful division of workload between the CPUs and FPGAs. Furthermore, various optimization techniques were used and precision, speed and energy consumption performance metrics were gathered. The results were compared with two previous implementations of the same localization problem using Exact Inference: A ProBT implementation (COTS software for Bayesian Programming and Inference) on a conventional CPU and a generic toolchain developed under the EU FET project BAMBI (Bottom-up Approaches to Machines dedicated to Bayesian Inference) using an unconventional computing approach on re-configurable hardware. In addition, we describe the impacts of OpenCL overhead on FPGA resource usage. The limitations on the official support of OpenCL from manufacturers and vendors encountered during implementation are analyzed. Finally, further work opportunities on this topic are proposed.

Keywords: **Bayesian Inference, OpenCL, Heterogeneous Computing, Top Down Approach, Re-configurable Logic, FPGAs**

Resumo

Nesta tese é analisada a implementação de Inferência Bayesiana em lógica reconfigurável (FPGAs) utilizando uma abordagem ¨Top Down¨. A partir de uma implementação genérica num CPU, e utilizando a linguagem de programação paralela OpenCL, implementámos as componentes de limitadas pela capacidade de processamento em FPGAs instaladas em aceleradores discretos. Um problema de localização genérico foi implementado numa plataforma de computação heterogénea contendo dois CPUs Intel Xeon E5 e quatro FPGAs Intel Stratix V. O sistema foi optimizado de forma a extrair a sua máxima capacidade de processamento através de um cuidadoso balanceamento de carga entre os CPUs e as FPGAs. Foram ainda implementadas várias técnicas de optimização e adquiridas métricas de velocidade, precisão e consumo de energia. Os resultados foram comparados com duas implementações prévias do mesmo problema de localização utilizando Inferência exacta: Uma implementação em ProBT (software comercial para Programação Bayesiana e Inferência) num CPU convencional e uma ¨toolchain¨ genérica desenvolvida no âmbito do projecto Europeu FET BAMBI (Bottom-up Approaches to Machines dedicated to Bayesian Inference) utilizando hardware dedicado não-convencional. Foram ainda analisados os impactos do OpenCL na utilização de recursos da FPGA. As várias limitações no suporte oficial a OpenCL dos fabricantes e comercializadores encontradas durante a implmentação foram analizadas. Por último, são propostas oportunidades de trabalho futuro sob este tópico.

Palavras-chave: **Inferência Bayesiana, OpenCL, Computação Heterogénea, Abordagem ¨Top Down¨, Lógica Reconfigurável, FPGA**

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

BAMBI	Bottom-up Approaches to Machines dedicated to Bayesian Inference
COTS	Commercial Off The Shelf Software
CPU	Central Processing Unit
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
HDL	Hardware Description Language
ITRS	International Technology Roadmap for Semiconductors
OpenCL	Open Computing Language
RICH	Ring-imaging Cherenkov detector

Chapter 1

Introduction

1.1 Motivation

From its beginning, the field of *robotics* has shifted from an automatic to an autonomous perspective. In its early days, the focus was the development automatic tools capable of reliably performing simple and repetitive tasks at high speeds. However, with advances in computing power, data acquisition, computer vision, hardware reliability and other related fields the objective slowly evolved into a quest to achieve autonomous operation. This autonomy now allowed for the appearance of autonomous vehicles, industrial and social robots.

One of the main challenges in the path for full robotic autonomy is perception. Early robots and computer-based devices were in a state of sensory deprivation [3]. With advances in sensory technologies, autonomous agents became overloaded with information. This led to the necessity to efficiently transform this raw data from an autonomous agent data acquisition systems into a useful and correct representation of its surroundings. In order to achieve this, agents must be able to efficiently deal with uncertainty.

To include the degree of belief into the computations, probabilistic alternatives to the traditional symbolic logic (commonly known as Boolean Algebra) have been successfully used. In the field of robotic perception Bayesian Inference is widely accepted and used. However, due to the cardinality of Bayesian Inference computations, even simple perception problems can easily become intractable, overloading traditional CPU based implementations.

It is a widely held belief among the scientific community that the miniaturization of processors is reaching its limit. According to the International Technology Roadmap for

Semiconductors (ITRS) [4], current generation manufacturing processes are able to reliably mass produce 14nm processors, with some foundries beginning pre-production runs of 10nm technology. It is is expected for this trend to reach its limit with the 5nm transistor, which will have a gate only 25 atoms wide. At this scale quantum effects and Strong and Weak forces assume a significant role which compromises the transistor operation. Despite this limitations, the introduction of some innovative manufacturing processes have allowed Moore´s Law to maintain its relevance. For example, the exploration of a third dimension of the silicon wafer called FinFET.

Current workloads both in scientific and enterprise computing markets require a great deal of perception. This requirement applies not only to large-scale high-performance servers but also to low-power domains due to the growing interest in cyber-physical systems including autonomously operating smart devices [5]. This growth in demand as created what is known as the *Compute Gap*. This is the difference between the future global necessities in computational power and the predicted availability.

In this context the European BAMBI project [6](Bottom-up Approaches to Machines dedicated to Bayesian Inference) was conceived. It aimed at tackling current technologies limitations by developing unconventional architectures to tackle Bayesian Inference problems. It aimed to create generic Bayesian Machines, based on stochastic computing, able to perform exact and approximate inference using less power, resources and still achieving satisfactory computing speed and precision. The original roadmap included the design and manufacturing of an easy-to-deploy Bayesian Application-Specific Integrated Circuit ((B)ASIC). However, due to the high non-recurring costs involved in ASIC production and the proposed architectures necessity to develop an entirely new chip for every change in the probabilistic model, this objective was abandoned in favor of implementation on Re-configurable Hardware (FPGAs)[7].

Throughout the implementation of a real world perception scenario using Inference architectures developed under BAMBI [8][9][7], some of the challenges commonly associated with Re-configurable Hardware development using Hardware Description Languages (HDL) were encountered. These challenges include the necessity of specialized developers, knowledgeable in low level development, hardware logic design and Register Transfer Level to design and verify complex designs and long compilation times. Furthermore, due to the lack of development standards across different Re-configurable Hardware platforms HDL design and optimizations are not fully platform/vendor independent, this implies maintenance costs for every change or upgrade in accelerator boards/FPGA chips[10]. In addition, integration with existing systems creates another set of challenges, requiring the development of the low level communication and control interfaces with peripheral devices or host processors.

To address the challenges in parallel programming on GPUs, the OpenCL (Open Computing Language) standard was created. Initially targeting GPUs, quickly evolved into a framework for writing programs that execute across CPUs, GPUs, FPGAs, DSPs and other processors, being best suited for computing in heterogeneous processing platforms. The standard includes a set of programming languages and APIs to control and execute parallel computations across the devices[1].

1.2 Objectives

In this work, we propose a different path in order to accomplish the same task - Fusion of six noisy sensors in order to determine the location of a boat in a grid. This task was proposed as a case study in [11] to measure the performance of the BM1 architecture developed under BAMBI. In [8] a PCIe interface with an host computer and a ROSBridge were developed.

We take a Top Down approach, implementing exact inference on conventional hardware (CPUs and FPGAs) using OpenCL.

Firstly, we expect to demonstrate the feasibility and efficiency of an OpenCL implementation of Bayesian Inference on FPGAs.

Secondly, we intend to explore the capabilities of the heterogeneous system, by optimizing the distribution of the computations across the available CPUs and FPGAs.

Finally, using the Boat Localization case study, we intend to compare the development burden and the final product efficiency of our approach with the ProBT implementation (COTS software for Bayesian Programming and Inference) on a conventional CPU [11] and with the BM1 architecture developed under BAMBI project running on the same FPGAs mini-cluster [8][11].

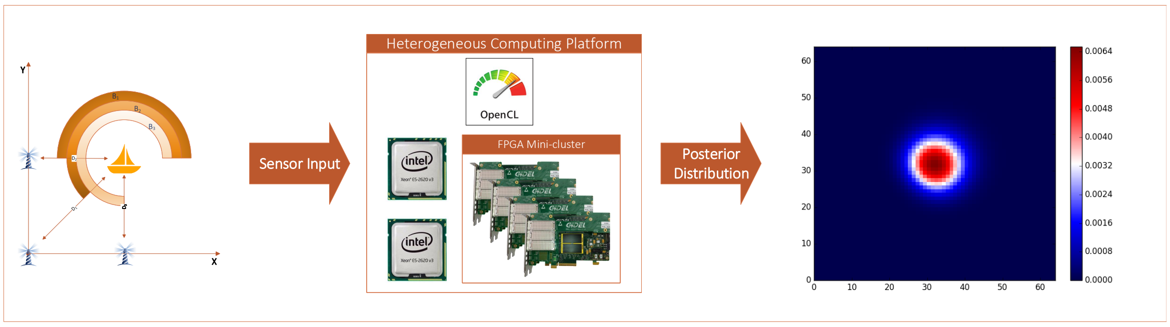


Fig. 1.1 System Overview.

1.3 Related Work

Zohouri et al. optimized OpenCL kernels for High Performance Computing with FPGAs. They analyzed FPGA and vendor specific optimizations. Furthermore, they compared OpenCL direct ports from GPU implementations with FPGA optimized versions[12].

At CERN a similar study is being conducted in preparation for the 2018 upgrade of the current LHC readout system [13]. A two socket platform from Intel with a Xeon CPU and a Stratix V FPGA communicating over QPI is being proposed as an experimental heterogeneous computing platform for computing the Cherenkov angle reconstruction of LHCb RICH particle identification algorithm. As part of the analysis, the current implementation in a Xeon processor is being compared with two heterogeneous CPU+FPGA implementations in both Verilog and OpenCL[14].

Wang et al. designed Melia - an OpenCL implementation of the MapReduce algorithm on FPGAs [15]. THey further developed a series of FPGA optimization techniques to improve the algorithm efficiency. Melia was benchmarked on Altera Stratix V GX FPGAs, both on single-FPGA and cluster settings, demonstrating the efficiency of the optimizations in comparison with equivalent implmentations on CPUs/GPUs.

Alves et al. perform a survey on computationalsolutions for Bayesian Inference [16]. Morales et al. compared GPU and FPGA implmentations of Binomial Option Pricing Using OpenCL[17].

At CHREC, development on OpenCL was compared with traditional HDLs for three image-processing algorithms[10] (Canny edge detector, Sobel filter, and SURF feature-extractor). Tests were performed on three different FPGA accelerator boards, one of which coincides with the ones used in our study.

At Karlsruhe, Weller et al. implemented and optimized partial differential equations (PDEs) solvers for scientific computation on accelerator FPGAs, achieving higher energy efficiencies than CPU only or CPU + GPU implementations [5].

Ferreira et al. demonstrate tractable implementation of exact Bayesian Inferece with a real-time OpenCL vision-based model to estimate gaze direction in a human-robot interaction (HRI) using GPUs [18].

On the other hand, on a Bottoms-Up Approach[7], unconventional hardware architectures are explored to perform the required inference. This was the course of action of BAMBI project: a generic toolchain was developed to generate unconventional hardware for probabilistic inference from stochastic building blocks, allowing bit level parallelism [7]. During this project, a Boat Location problem was used as a case study [11]. Later, Mira continued the work, integrating ROS and adding a PCIe interface to extract the full capabilities of the FPGA mini-cluster [8].

1.4 Key Contributions

- OpenCL implementation of Bayesian Inference on an heterogeneous computing system (CPU and FPGA) using a Top-down approach;
- Optimization of the inference kernel;
- Comparison of energy consumption, resources usage, computation speed and development effort with previous implementations using the Boat Location problem as benchmark;
- Trade-offs analysis for a Top-down approach using OpenCL.

1.5 Dissertation Overview

- **chapter 1** describes the motivation and objectives of this work. Furthermore, a summary review of related work in the field is given, and the key contributions of this work listed.

- **chapter 2** describes the theoretical foundations of this work, split twofold: Bayesian Inference mathematics and OpenCL framework.
- In **chapter 3** the implementation is described.
- **chapter 4** analyzes the results and compares them with two previous implementations of the same localization problem using Exact Inference: A ProBT implementation on a conventional CPU and an implementation produced by the generic toolchain developed under BAMBI project using the same Stratix V FPGA mini-cluster.
- In **chapter 5**, conclusions are drawn from the work under scrutiny and further research opportunities are proposed.

Chapter 2

Background

2.1 Bayesian Inference

Scientists have often turned to nature for inspiration. This is specially relevant in the field of perception applied to robotics, where the ability to capture and extract useful information from the environment is of paramount importance. Animals have always had the ability to perform decisions using incomplete and uncertain information regarding its surroundings. This probabilistic behaviour as long captured the interest of the scientific community in order to further improve perception algorithms. In the last couple of decades, Bayesian inference has permeated into most fields of scientific research. From social sciences and humanities all the way to medicine and biology, Bayesian inference is viewed as an alternative to conventional frequentist inference techniques statistical data analysis [3].

Bayesian Inference has become an important tool for modeling cognition and reasoning of both living organisms in fields such as Neuroscience and to perform artificial perception in fields such as Artificial Intelligence or robotics. However, Bayesian inference in these fields has always been limited by the computational requirements for real time implementations. Bayesian programs generally include a large number of inter-dependent random variables. Even considering that all variables have been discretized (analog to digital conversion of sensor input, for example), the complexity of the Bayesian program, specifically the cardinality of each random variable, produces a tractability issue. In other words, the computations are exceedingly resource and time consuming in conventional computing units for the inference to be performed in a reasonable amount of time.

Bayesian inference can be formally defined as the process of answering the question P(Searched | Known), or in other words, determining the posterior, by computing:

$$P(Searched|Known) \propto \sum_{Free} P(Searched \wedge Known \wedge Free) \tag{2.1}$$

Exact Inference

Theoretically, the intuitive method of performing inference is by computing the posterior distribution as accurately as allowed by the precision of the underlying hardware, reaching the exact values of the probabilities involved. When the computation of the posterior is feasible using a closed-form method, inference is said to be exact. Such an exhaustive method is not feasible for for the majority of cases, in fact it is only applicable in two scenarios: when all free variables are discrete (hidden Markov models) or when all distributions are linear and normal (Kalman filters). Event then, inference may be theoretically possible, but not computationally tractable. Due to the curse of dimensionality, if the cardinality of each random variable is too high, the computation may be excessively long for any real world application. In addition, if the program is exceedingly complex, an analytic solution may not even exist. Therefore, exact inference algorithms simplify the model computational tree using tree-based and graphical methods,in order to reduce the computational burden of the exhaustive computation of the posterior distribution probabilities.

Approximate Inference

When exact inference is not feasible, scientists trade computational precision and exhaustiveness for tractability. Using various methods and algorithms, the model is simplified and an approximate description of the posterior distribution is used, enabling the execution of the required inference. This is known as approximate inference.

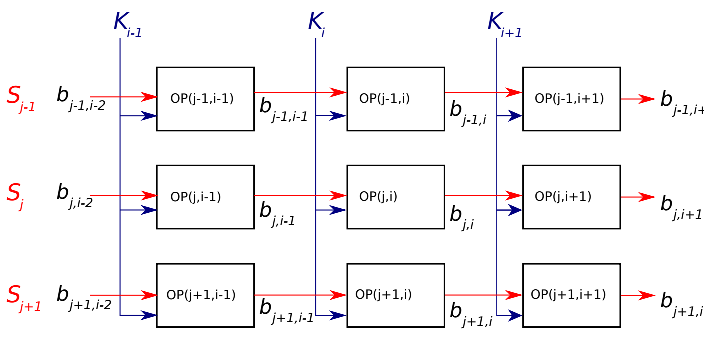
2.1.1 Bottom-up Approach - BAMBI project

In order to circumvent current limitations of real world implementations of Bayesian Inference, the European BAMBI project (Bottom-up Approaches to Machines dedicated to Bayesian Inference) was conceived. It followed a bottom-up approach, attempting to design unconventional hardware for probabilistic computation from new stochastic building blocks, effectively reshaping current inference models on a deep level. The project took the Bayesian paradigm to the hardware level - instead of attempting to perform Bayesian inference on conventional Boolean logic gates, it developed its Bayesian counterpart - the Generic Bayesian Gate, also known as GUT. Using this generic gate as a building block, various computing architectures were developed, that harnessed the power of stochastic computing to efficiently compute exact samples at the bit level. This successfully reduced the complexity of the required inference, increased the computation speed, and increased the energy efficiency, while maintaining the same level of accuracy as conventional methods. The original roadmap included the design and manufacturing of an easy-to-deploy Bayesian Application-Specific Integrated Circuit ((B)ASIC). However, due to the high non-recurring costs involved in ASIC production and the proposed architectures necessity to develop an entirely new chip for every change in the probabilistic model, this objective was abandoned in favor of implementation on Re-configurable Hardware (FPGAs)[7].

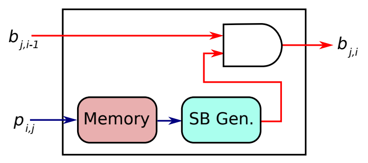
During the project, a compilation toolchain was developed to implement various computing architectures in re-configurable logic. The generic architecture that performed exact inference is called the Bayesian Machine 1 (BM1). The BM1 is a massively parallel Bayesian fusion machine, which requires all known variables to be considered conditionally independent between each other for any searched variable. It computes the posterior probability distribution on the searched variable S, using the knowledge of the prior distribution $P(S)$ and the set of conditional distributions $P(K_i|S)$, where Z is a normalization constant:

$$P(S|K_n) = \frac{1}{Z}P(S)\prod_{i=1}^{N}P(k_i|S) \propto P(S)\prod_{i=1}^{N}P(k_i|S) \tag{2.2}$$

From a Bayesian Program defined in ProBT®, the computation tree is extracted and a set of probability distribution Look-up Tables (LUTs) are computed. Using this input, the toolchain synthesizes the necessary components for the FPGAs, reconfigured them using the previously generated bit-stream. Additionally the memory is preloaded with the prior distribution $P(S)$ vector and with the LUTs containing the set of conditional distributions $P(K_i|S)$. The inference is performed and the results stored on the host machine.



(a) BAMBI´s BM1 architecture.



(b) BAMBI´s BM1 element.

2.1.2 Top-down approach

In this work a top-down approach was followed, by implementing an exact inference computation tree on an FPGA mini-cluster, using the OpenCL framework. In a top-down approach, the inference computations are implemented on the underlying hardware using standard components, performing regular mathematical and logical operations. These mathematical operations are evaluated using either LUTs (Lookup Tables) computed in advance, or using low precision representations of the probabilities.

The model is defined using the ProBT ProBT® generic inference programming tool from the Bayesian programming formalism [19]. Using the Successive Reduction Algorithm (SRA), ProBT produces a simplified computation tree, which is then implemented on the FPGA mini-cluster using OpenCL.

2.1.3 Bayesian Programming

In order to define and compare Bayesian inference problems, a common language was needed. The required formalism appeared with the definition of the Bayesian Program (BP). This generic language allows researchers to build probabilistic models and later compute decision and inference problems on those same models. A Bayesian Program consists of two parts:

Description - Probabilistic model of the physical occurrence or behaviour being modelled;

- **Specification** - Exhaustive enumeration of the programmer's formal knowledge of the model;
- **Identification** - Learning procedure to estimate the model's free parameters from an experimental data set;

Question - specifies the inference problem to be computed using this model.

All available knowledge of the physical occurrence being modelled is encoded in the joint probability distribution. However, as described in section 2.1, this joint distribution usually causes tractability issues due to its complexity. Therefore, the specification is used to detail a tractable method to compute the joint distribution. The Identification provides the learning methods to estimate the values of the model's free parameters from the observed data set. Finally, the Question is obtained by dividing the relevant variables into three sets: the searched variables, the known variables and the free variables. The variables *Searched*, *Known* and *Free* denote the conjunction of variables belonging to the aforementioned sets.

For each value of the variable *Known*, a question is defined as $P(Searched|Known \wedge \pi)$.

$$\textbf{BN()=}\begin{cases} \textbf{Description}\begin{cases} \textbf{Specification}\begin{cases} \textbf{Variables} \\ \textbf{Decomposition} \\ \textbf{Parametric Form} \end{cases} \\ \textbf{Identification} \end{cases} \\ \textbf{Question} \end{cases} \tag{2.3}$$

2.1.4 ProBT® programming tool

Various probabilistic programming languages are currently available to the scientific community, both as Commercial Off The Shelf Software(COTS) or as Open-Source source software. These generally include and Application Programming Interface (API) to allow the specification of the model and an inference engine that implements various algorithms in order to solve the inference problems proposed.

In this work, ProBT® was used to define the case study model and to generate the computational tree. This COTS was selected for compatibility reasons, given that it was the software used throughout BAMBI project.

ProBT®, developed and maintained by ProBAYES is a generic inference programming tool that facilitates the creation of Bayesian models and their re-usability using the Bayesian programming formalism [19]. This library is a powerful programming tool for low-to-medium complexity models consisting of two layers:

ProBT Engine - set of high-performance inference algorithm modules developed in C++;

ProBT API - an Application Programming Interface available in C++ and Python for accessing the ProBT Engine functions.

After model specification, ProBT® generates a computation tree which is simplified using the Successive Reduction Algorithm (SRA)[20]. The simplified computation tree is then stored in a file to be later retrieved by the OpenCL host for implementation on the FPGA mini-cluster.

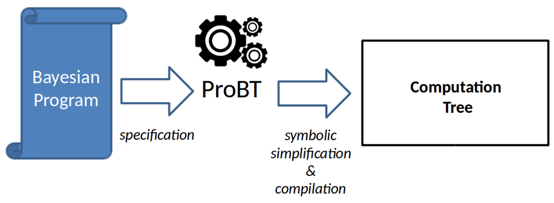


Fig. 2.2 ProBT Compilation Toolchain

2.1.5 Bayesian Networks

Bayesian inference usually requires complex formalizations and algebraic manipulations, decreasing the readability of the Bayesian model. To address this issue, graphical representations were developed, improving the readability and clarity of the models. One of the most popular graphical representations are Bayesian Networks (BNs). Those directed acyclic graphs (DAGs) represent variables as nodes and the dependencies between variables as arcs. The completed network fully characterizes the Bayesian model, allowing the extraction of all necessary information in order to perform Bayesian inference.

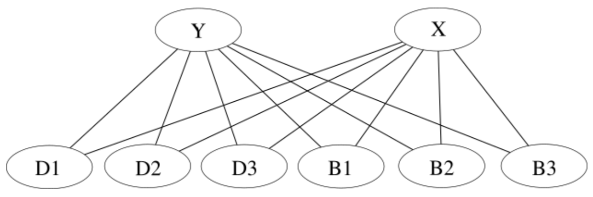


Fig. 2.3 Bayesian Network of Boat Localization Example

2.1.6 FPGA vs CPU

Nowadays, the *de facto* technology for re-configurable logic in the computing world are Field Programmable Gate Arrays(FPGAs). Over the last decade, advances on both the hardware and software side have increased the adoption rate of this technology and introduced it to new applications and workloads. With the slowdown in CPU scaling and the ending of Moore's Law [4], alternatives to CPUs have become more attractive. Following a path similar to that of GPUs, FPGAs are being increasingly deployed in novel configurations, as part of heterogeneous computing systems. In the next paragraphs we proceed to review some of the advantages and disadvantages of FPGAs for computationally intensive tasks.

On the one hand, due to its re-configurability, FPGAs are highly flexible. On FPGAs only the necessary hardware is instantiated, reducing the design critical path and therefore increasing the processing speed by increasing the clock frequency[21]. In addition, unused hardware doesn't consume energy, contrary to CPUs, leading to higher energy efficiency[22]. This flexible implementation also allows gains with massive paralelization, by enabling a denser instantiation of computing blocks. On the other hand, the most attractive point for FPGAs is also their biggest hurdle. Development at such a low level is tremendously complex, requiring specialized Hardware Developers and slowing the development cycle. In order to unleash the full potential of the hardware platform without exposing the software developers to all of the gritty details of the underlying hardware, an effort as been made by manufacturers to move from low level Hardware Description Languages such as Verilog and VHDL to higher level abstractions, such as OpenCL. With these High level Synthesis (HLS) tools hardware can be built with fewer lines of code.

There is an inherent tradeoff when using FPGAs. The flexibility and re-configurability, come at the expense of the higher efficiency of Dedicated Circuits such as ASICs.

For FPGAs, recently there have been major efforts from technology leaders to better integrate FPGA accelerators within data center servers (e.g., Microsoft Catapult, IBM CAPI, Intel Xeon+FPGA project) [23] for both commercial and scientific High Performance Computing (HPC) workloads.

Microsoft Catapult took a leap forward by installing over 100.000 FPGAs on their data centers since 2015 using a novel architecture. Each conventional server has an FPGA installed on an accelerator board. The board is connected to the server through PCIe and through the Network Interface Card (NIC), effectively bypassing the network traffic between the Top Of Rack Switch and the NIC. PCIe is used for traditional compute offload tasks and for FPGA reconfiguration. However, as the FPGAs are directly connected to the network, can share the workload with underused neighbouring FPGAs bypassing the server altogether[24].

In Germany's Paderborn University the first phase of the Euro Noctua cluster project is underway [25]. This cluster dedicated to scientific research will contain 32 Intel Stratix 10 FPGAs for early experiments porting, programming, scaling, and understanding how some traditional (and non-traditional including K means, image processing, machine learning) HPC applications respond to an FPGA boost.

Some workloads ready for FPGA acceleration are specific solvers for computational chemistry, nanostructure materials and electromagnetics as well as analysis of electron structure where most of the operations are performed on large matrices.

2.2 Open Computing Language (OpenCL)

”*OpenCL (Open Computing Language) is an open royalty-free standard for general purpose parallel programming across CPUs, GPUs and other processors, giving software developers portable and efficient access to the power of these heterogeneous processing platforms. OpenCL supports a wide range of applications, ranging from embedded and consumer software to HPC solutions, through a low-level, high-performance, portable abstraction. By creating an efficient, close-to-the-metal programming interface, OpenCL will form the foundation layer of a parallel computing ecosystem of platform-independent tools, middleware and applications. OpenCL is particularly suited to play an increasingly significant role in emerging interactive graphics applications that combine general parallel compute algorithms with graphics rendering pipelines.*”

A. Munshi, “OpenCL 1.2 Specification”, 2012

2.2.1 Origins

OpenCL was initially developed by Apple in order to provide an open interface for any application to massively parallel computing power of GPUs. Before its development GPUs were only accessible to graphics applications or through limited proprietary interfaces. The initial version of the specification was released in August 2009 and as of May 2017 the current version is 2.2.

The specification is currently maintained by the Khronos Group - a not for profit, member-funded consortium. The consortium develops and maintains free open standards for 3D graphics, Virtual and Augmented Reality, Parallel Computing, Neural Networks, and Vision Processing on a wide variety of platforms and devices from the desktop to embedded and safety critical devices. The list of consortium members dedicated to the development of the OpenCL standard include some of the biggest manufacturers of computing hardware in the world, such as: Qualcomm, ARM, Apple, Intel (which has bought Altera), Xilinx, NVIDIA Corporation,STMicroelectronics, IBM Corporation, Samsung Electronics, IBM and AMD.

2.2.2 Architecture

Initially targeted to GPUs, OpenCL is currently a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators.

The OpenCL specification includes programming languages based on C99 and C++11 for programming the devices, an application programming interface (API), libraries and a runtime system to control the platform and execute programs on the compute devices.

The target of OpenCL are expert programmers wanting to write portable yet efficient code. Therefore OpenCL provides a low-level hardware abstraction plus a framework to support programming. Many details of the underlying hardware are exposed to the developer.

The OpenCL framework can be described using a hierarchy of models:

- Platform Model
- Memory Model
- Execution Model
- Programming Model

Platform Model

The OpenCL heterogeneous computing platform is divided in a host (CPU), connected to one or more OpenCL Compute Devices (CPUs, GPUs, DSPs, FPGAs,...). Each device is divided into one or more Compute Units (CUs), which in turn are divided into multiple Processing Elements (PEs).

The developed program runs on the host, which in turn issues commands to the Processing Elements within each device to control the execution. Instructions within a compute unit can be executed in a Single Instruction Multiple Data (SIMD) or in a Single Program Multiple Data (SPMD).

Execution Model

During runtime, execution is divided in two: a host program running on the host and kernels executing on the OpenCL devices. The host program controls the kernels context and man-

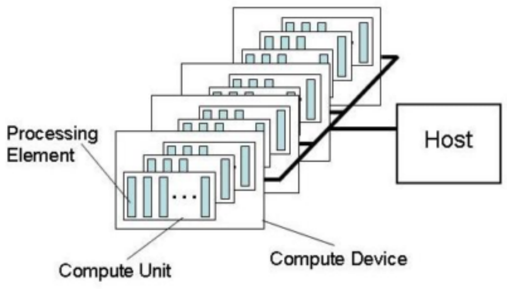


Fig. 2.4 OpenCL Platform Model. [1]

ages their execution.

When a kernel is queued for execution by the host, its context is created and an instance is created on the OpenCL device. Each instance is called a *work-item* and is a completely independent instance of the kernel code and is identified by a global ID. Work-items are organized into *work-groups*. Each work-group has a work-group ID and each work-item is assigned a unique local identifier within each work-group. The global identifiers index space is called an NDRange. The NDRange can have one, two or three dimensions .

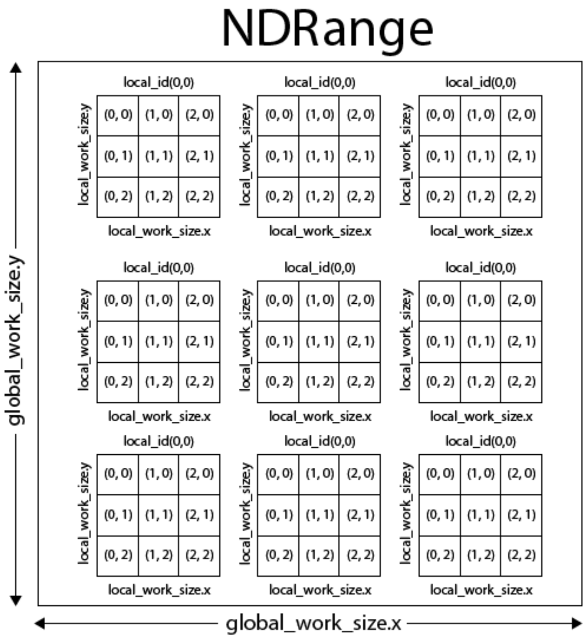


Fig. 2.5 NDRange kernel index space. [1]

Memory Model

Work-items executing a kernel have access to four distinct memory regions:

- **Global Memory** - allows read/write access to all work-items in all work-groups;
- **Constant Memory** - A subset of global memory that remains constant during kernel execution. Only the host can allocate and initialize constant memory;
- **Local Memory** - Memory region local to a work-group. It allows sharing of variables across all work-items of the same work-group;
- **Private Memory** - Memory private to each work-item.

	Global	Constant	Local	Private
Host	Dynamic allocation Read / Write access	Dynamic allocation Read / Write access	Dynamic allocation No access	No allocation No access
Kernel	No allocation Read / Write access	Static allocation Read-only access	Static allocation Read / Write access	Static allocation Read / Write access

Table 2.1 OpenCL - Memory Allocation and Memory Access by Device

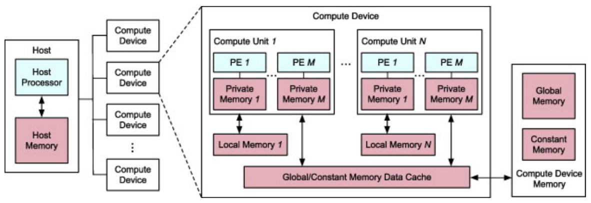


Fig. 2.6 OpenCL Device Memory Architecture [2].

Programming Model

OpenCL explicitly supports two programming models: the data parallel and the task parallel programming model. The primary model for the OpenCL framework is the data parallel model.

In the data parallel programming model, the computation is defined as a sequence o instructions performed to multiple data elements from a memory object. In the task parallel model each kernel instance is independent of of its surroundings. This is equivalent to executing a the kernel on a work-group with a single work-item.

2.2.3 Re-configurable Systems Support

OpenCL support for FPGAs was major step in FPGA ease of deployment. The OpenCL framework provides a kind of a modular and parameterizable soft multicore processor architecture, simplifying workload acceleration using FPGAs.

The two major manufacturers of FPGAs, Intel FPGA (former Altera) and Xilinx are currently active members of the consortium developing the OpenCL specification. In addition, both Intel and Xilinx, currently provide OpenCL development support for FPGAs through Altera OpenCL Software Development Kit (SDK) and through Xilinx SDAccel Development Environment [26] [27].

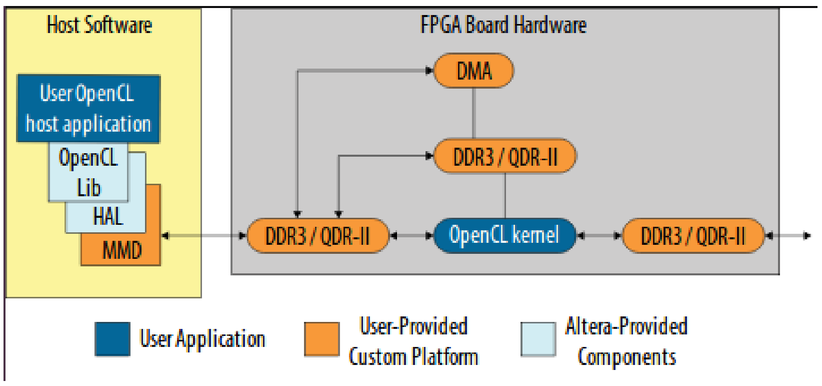


Fig. 2.7 Altera OpenCL SDK Software Architecture.

Intel FPGA OpenCL SDK is divided in two components: the **Offline Compiler** and the **OpenCL Runtime Environment**.
The offline compiler is controlled through a command line interface. It is responsible for the compilation of the OpenCL kernels. In addition it provides a variety of tools to help the developer produce more efficient kernels. The intermediate compilation provides syntactic error checks and resource usage estimation, without building the hardware. This accelerates development iterations, as this intermediate step only takes a couple of minutes. In addition, the offline compiler supports emulation, allowing functionality testing before building the hardware. Finally, it contains a powerful profiler, capable of automatically deploying performance probes and counters into the kernel program. During execution on the FPGA, the profiler collects performance data which is then automatically compiled into a user friendly interactive report.

The Intel OpenCL Runtime Environment is the software responsible to interface with the accelerator devices. To achieve this task, the user generated host program is run on top of the OpenCL API libraries. This libraries interface with the Intel FPGA Hardware

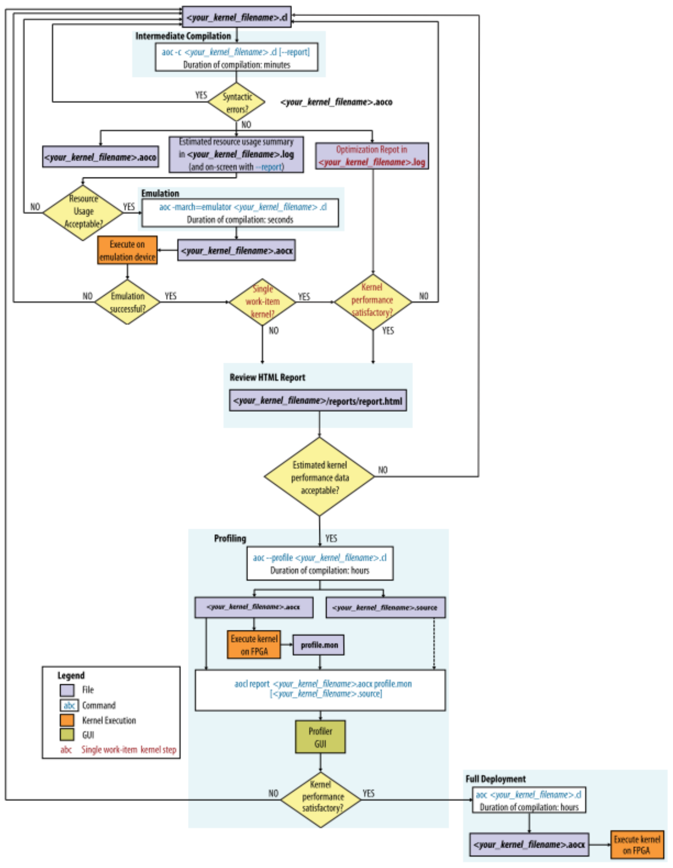

<your_kernel_filename>.cl
Intermediate Compilation
aoc -c <your_kernel_filename>.cl [--report]
Duration of compilation: minutes
Syntactic errors?
YES
NO
<your_kernel_filename>.aoco
<your_kernel_filename>.aoco
Estimated resource usage summary in <your_kernel_filename>.log (and on-screen with --report)
Optimization Repot in <your_kernel_filename>.log
Resource Usage Acceptable?
NO
YES
Emulation
aoc -march=emulator <your_kernel_filename>.cl
Duration of compilation: seconds
<your_kernel_filename>.aocx
Execute on emulation device
Emulation successful?
NO
YES
Single work-item kernel?
YES
NO
Kernel performance satisfactory?
NO
YES
Review HTML Report
<your_kernel_filename>/reports/report.html
Estimated kernel performance data acceptable?
NO
YES
Profiling
aoc --profile <your_kernel_filename>.cl
Duration of compilation: hours
<your_kernel_filename>.aocx
<your_kernel_filename>.source
Execute kernel on FPGA
profile.mon
aocl report <your_kernel_filename>.aocx profile.mon [<your_kernel_filename>.source]
Profiler GUI
Kernel performance satisfactory?
NO
YES
Full Deployment
aoc <your_kernel_filename>.cl
Duration of compilation: hours
<your_kernel_filename>.aocx
Execute kernel on FPGA
Legend
File
abc Command
Kernel Execution
GUI
abc Single work-item kernel step

Fig. 2.8 Altera OpenCL Design Flow.

Abstraction layer (HAL). This layer provides the interface with third party Memory Mapped Device (MMD) layer. This last layer is part of the Board Support Package(BSP) provided by accelerator boards manufacturers. The BSP contains all of the board information and drivers required for kernel compilation and host interfacing. The MMD layer interfaces directly with the communication drivers, such as PCIe, in order to control the accelerator board.

2.3 FPGA Reconfiguration Protocols

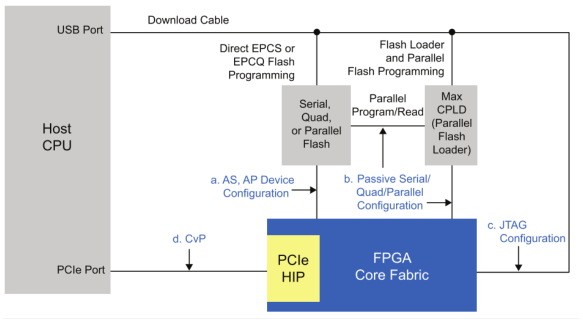

USB Port
Download Cable
Host CPU
Direct EPCS or EPCQ Flash Programming
Flash Loader and Parallel Flash Programming
Serial, Quad, or Parallel Flash
Parallel Program/Read
Max CPLD (Parallel Flash Loader)
a. AS, AP Device Configuration
b. Passive Serial/ Quad/Parallel Configuration
c. JTAG Configuration
d. CvP
PCIe Port
PCIe HIP
FPGA Core Fabric

Fig. 2.9 Block Diagram - Device Reconfiguration Mode.

2.3.1 Configuration Via Protocol (CvP)

A CvP system consists of an FPGA and a PCIe host. The FPGA contains at least one PCIe Hard IP block for CvP and other PCIe applications. In CvP the design is partitioned into two images: core image and periphery image. The periphery image is static and cannot be reconfigured. The core image consists of a reconfigurable region that can be programmed during runtime. CvP supports a multiple endpoints topology, providing the flexibility of selectively programming any number of FPGAs connected to the host through PCIe.

2.3.2 JTAG (Joint Test Action Group)

JTAG (Joint Test Action Group) is the industry standard protocol for debugging and verification of hardware designs. It specifies the use of a dedicated debug port implementing a serial communications interface. Throught this serial interface, FPGAs can be reprogrammed.

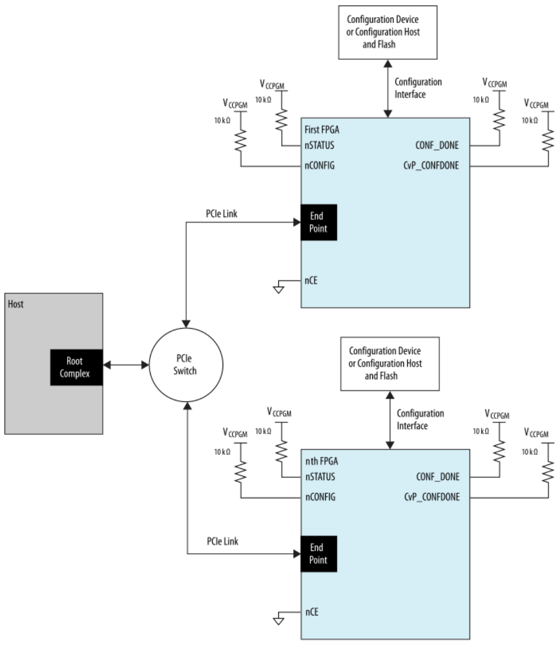


Fig. 2.10 Block Diagram - Multiple Endpoint Configuration Via Protocol.

Chapter 3

Implementation

3.1 Experimental Setup

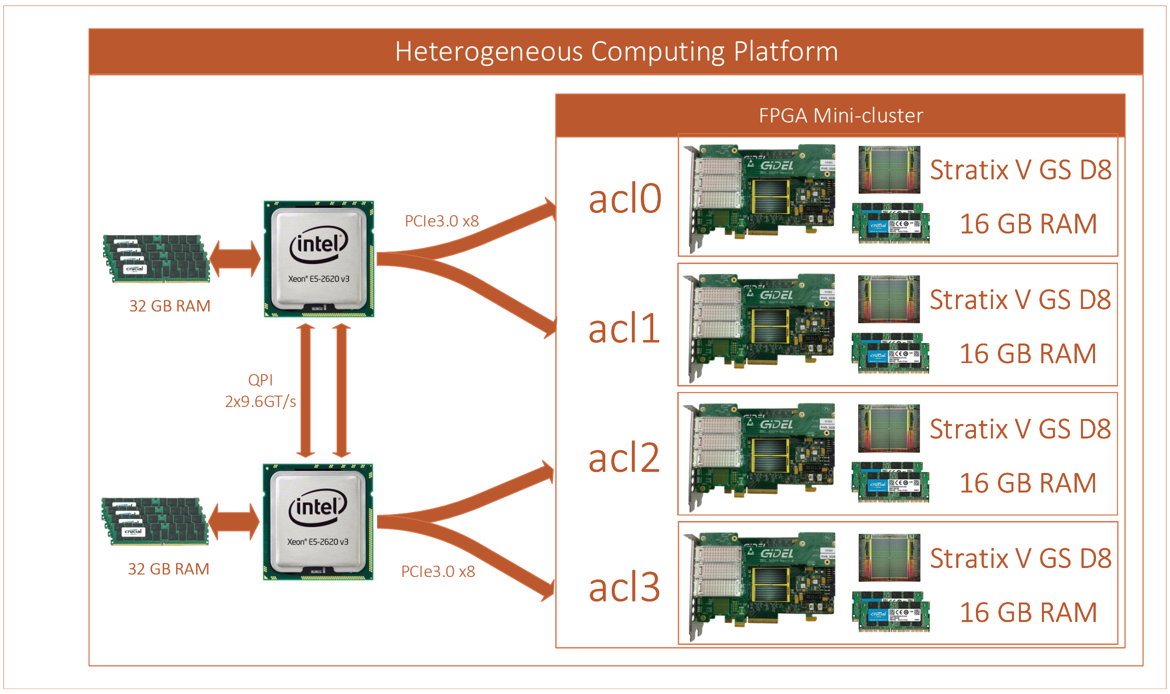


Fig. 3.1 Physical System Diagram.

In order to develop the work presented in this dissertation, an heterogeneous computing platform was used. This platform was acquired under BAMBI project in 2015. The rationale behind its hardware specifications was to replicate Florida´s NSF-CHREC recently deployed Novo-G# High Performance Computer (HPC)[28]. This would enable the integration into the Novo-G# Forum. The system comprises two Intel Xeon CPUs, each with 12 cores and four Intel Stratix V GS D8 FPGAs installed in Gidel´s ProceV accelerator boards.

Due to incompatible libraries versions between the ProBT software and the operating system version (CentOS 6.9) required by the older ProceV boards, a conventional laptop with Ubuntu 16.04 was used to generate the inference model and to simplify the computation tree.

In the early stages of development, a Terasic DE1-Soc development kit was used with the expectation to provide faster development cycles. However, the combined dimension of the inference computations and the overhead of OpenCL easily exceeded this FPGA size.

3.1.1 Base Platform

A Supermicro 4U server case was used. The requirements in terms of power, airflow and size for the FPGA accelerator boards installation are identical to those of server grade GPUs, for which the case was designed. The system was initially built with only 16GB of RAM, which proved insufficient for hardware synthesis tasks 3.2.2. During this dissertation, it was upgraded to the currently installed 64GB. The system has two Intel Xeon E5-2620 CPUs running at 2.6GHz, each with 12 cores. The CPUs are tightly interconnected through two Intel QuickPath Interconnect (QPI) interfaces, supporting 9.6GT/s transfer speeds each.

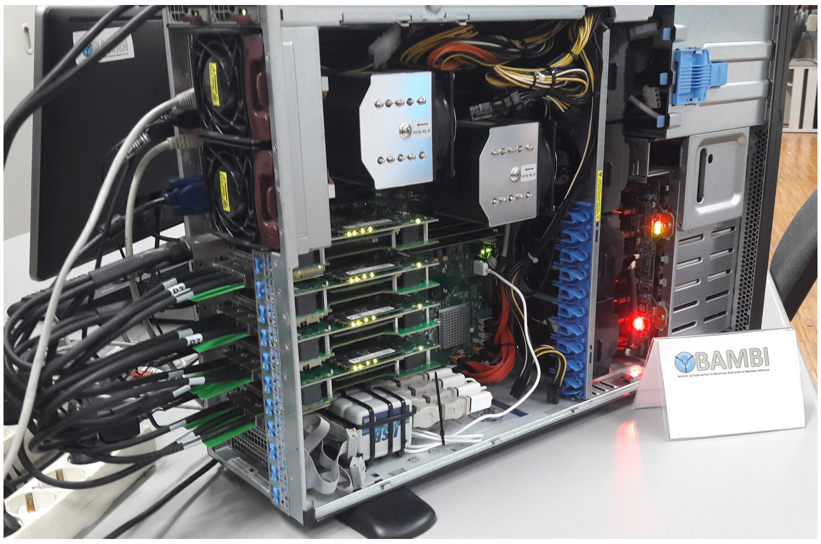


Fig. 3.2 Heterogeneous Computing Platform - Visible four Gidel ProceV boards connected in a 3D torus using fiber-optical cables. Beneath the top coolers, two Intel Xeon E5-2620 v3 CPUs.

The most relevant platform specifications for the proposed workload are:

- SuperMicro SC747TQ-R1620B Server Chassis
- CentOS 6.9 64bits
- 64 GB RDIMM ECCR DDR4 2133MHz (16x4GB)
- 2x Intel Xeon CPU E5-2620 v3 @ 2.40GHz (Total: 24 CPU cores)
- 256 SSD
- 4TB HDD
- 11x Full-Height, Full-Length PCIe, 8 lanes Expansion Slots Optimized for 4x Double Width Accelerator boards
- 4x 9cm PWM Cooling Fans & 2x 8cm Rear PWM Fans
- 4U Full Tower Chassis
- 1620W Redundant Power Supply
- 8x 3.5" Hot-Swappable HDD Drives
- IPMI interface for remote control

3.1.2 FPGA Mini-cluster

The FPGA cluster comprises four high-end Intel Stratix V GS D8 FPGAs installed on four Gidel ProceV accelerator cards. As per the manufacturer requirements, each card has two SODIMM slots fully populated with 16GB of RAM. In addition, four PHS_3QSFP daughter-boards are installed. The combined bandwith of the three QSFP+ and the CXP connectors amounts to 2400Gbit/s full duplex for each of the four cards.

The cards are divided in two pairs, each connected to a different CPU through a PCIe 3.0 bus. However, due to the CPUs QPI interconnects, the four boards are able to communicate with each other. In addition, a 3D torus network is installed to connect the four boards. This topology is able to fully use the 2400Gbit/s full duplex bandwidth available in each FPGA. Therefore, bandwidth available between any two boards in the cluster is 800Gbit/s full duplex.

Stratix V FPGAs fully support OpenCL. However, ProceV boards did not support OpenCL natively. This feature as only added in the end of 2014.

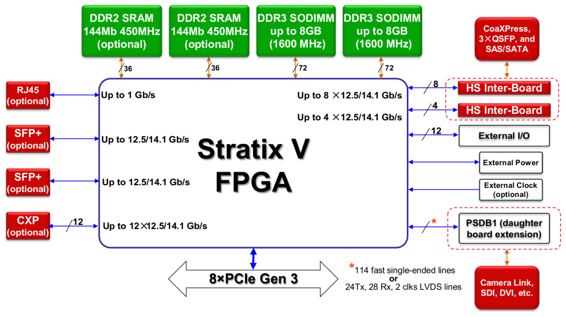
In order to reconfigure the FPGAs two protocols can be used: Configuration Via Protocol (CvP) or JTAG. The CvP method uses Stratix V Hard IP to reconfigure over PCIe while the JTAG method requires an Intel USB Blaster connected to the JTAG port on the board.

Each ProceV board contains:

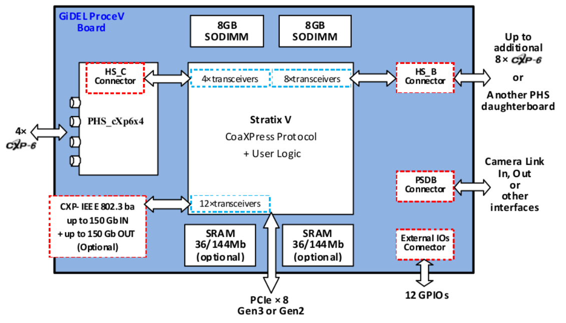
- Intel Statix V GS D8 (5SGSMD8K2F40)

(a) Gidel´s ProceV board.

(b) Gidel´s ProceV board with PHS_3QSFP daughterboard.


DDR2 SRAM 144Mb 450MHz (optional)
DDR2 SRAM 144Mb 450MHz (optional)
DDR3 SODIMM up to 8GB (1600 MHz)
DDR3 SODIMM up to 8GB (1600 MHz)
CoaXPress, 3×QSFP, and SAS/SATA
36
36
72
72
RJ45 (optional)
SFP+ (optional)
SFP+ (optional)
CXP (optional)
12
Up to 1 Gb/s
Up to 12.5/14.1 Gb/s
Up to 12.5/14.1 Gb/s
Up to 12×12.5/14.1 Gb/s
Stratix V FPGA
Up to 8 ×12.5/14.1 Gb/s
Up to 4 ×12.5/14.1 Gb/s
8
4
12
HS Inter-Board
HS Inter-Board
External I/O
External Power
External Clock (optional)
PSDB1 (daughter board extension)
8×PCIe Gen 3
*114 fast single-ended lines or 24Tx, 28 Rx, 2 clks LVDS lines
Camera Link, SDI, DVI, etc.

(c) Gidel´s ProceV Board block diagram.



(d) Proc High Speed (PHS_3QSFP) daughter-board block diagram.

Fig. 3.4 Altera USB Blaster - Used to reconfigure the FPGAs through JTAG.

  - 695K Logic Elements (LE)
  - 262400 Adaptive Logic Module (ALMs)
  - 1050 registers
  - 36x 14.10Gbit/s transceivers
  - 28 Fractional (PLLs)
  - 2567 M20K Memory Blocks
  - 3926 Variable Precision Multipliers (18x18)
  - 1963 Variable Precision Multipliers (27x27)
  - 6 DDR3 SDRAM x72 DIMM interfaces
- 1 CXP connector compatible with 100Gbit/s or 3x40Gbit/s Full Duplex
- 2 SFP+ suitable for 10GBit/s Full Duplex
- 1 Gidel Proc High Speed (PHS_3QSFP) daughterboard with 3x QSFP+ connectors, each supporting 40Gbit/s Full Duplex
- 2 DDR3 ECC SODIMMs RAM Banks with 16GB installed
- Terasic USB Blaster Cable P0302

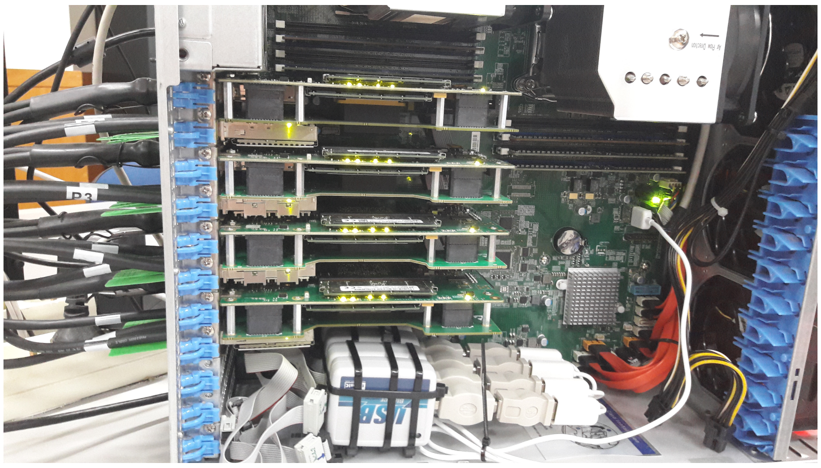


Fig. 3.5 Four Gidel ProceV boards connected to four Terasic USB Blasters through JTAG.

3.2 Preliminary Approaches

3.2.1 Altera DE1Soc Board

In the early stages of development, a DE1Soc kit was tested. The kit combined a Cyclone V and a ARM Cortex A9 processor in a system-on-chip (SOC). This board allowed fast iterations, with the compilation of kernels with under 50% resource usage taking an average of 45 minutes. However, the Cyclone V fabric proved too small for the proposed work. Due to OpenCL overhead, the largest boat localization kernel that we were able to fit on this FPGA had a cardinality of 16, which corresponds to a grid of 4x4. This board was abandoned in favor of the FPGA Mini-cluster.

3.2.2 Hardware constraints

Throughout development, various software and hardware limitations and incompatibilities were discovered. The diagnostics, resolution or mitigation of this problems severely delayed the dissertation timeline. Some of the most severe limitations are listed below:

Gidel´s Board Support Package only supported version 14.1 of the Intel FPGA OpenCL SDK. However, version 14.1 of the SDK Offline Compiler doesn´t support the *–high-effort* flag. Without the use of this flag, the compilation of large designs fails during the hardware generation stage because it fails to meet fitting constraints.

Gidel´s BSP is poorly documented and includes various bugs. In addition, being a commercial product, the source code is not available for inspection or correction, leading to the impossibility to correct the problems encountered. The biggest limitation is the fact that the PCIe drivers are unstable. It was repeatedly observed that whenever an attempt to reconfigure any of the FPGAs after more than a couple of hours since the previous reconfiguration, the PCIe driver generates an error, freezes the server and forces it to restart. After this reboot, each FPGA has to be reconfigured through JTAG and the server rebooted again. This process took approximately 25 minutes, severely delaying the development.

It was also discovered that the selection of the device being programmed in a JTAG chain was hard coded in the Board Support Package. This required the acquisition of three additional USB Blasters in order to be able to reprogram the for boards through JTAG without having to phisically disconnect and reconnect the USB Blaster.

As the boards support CvP (reconfiguration through PCIe), JTAG should only be required for the first reconfiguration after a system failure or after a hardware change. However, random fallbacks from CvP to JTAG were observed regularly, increasing the reprogramming time significantly.

The initial hardware configuration did not comply with Intel OpenCL SDK minimum RAM requirements for Stratix V development. Attempts to compile low complexity kernels with only 16GB of RAM resulted in aborted compilations. An upgrade was made to 64GB.

3.3 Case Study: Boat Localization

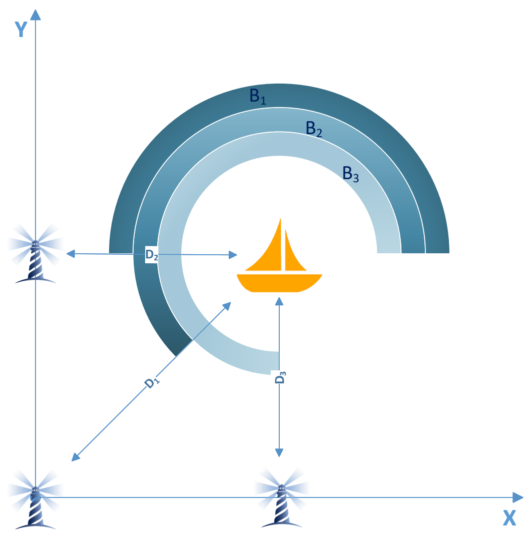


Fig. 3.6 Boat Localization problem - Three beacons are visible from the boat. The boat is able to detect its distance (D_1, D_2, D_3) and bearing (B_1, B_2, B_3) to each beacon.

One of this thesis objectives is the comparison of the OpenCL Bayesian Inference implementation of a well known Boat Localization problem with a conventional CPU approach and with the Bayesian Machines previously developed during BAMBI [11, 7].

The problem consists in determining the position of a boat in a discrete grid of 64x64 cells (4096 possible positions). We consider the existence of three landmarks at the edge of the grid, according to figure 3.6.

Aboard the boat, three distance sensors and three bearing sensors capture the distance and bearing of the boat to each of the three landmarks. We consider that the sensors signal is affected by noise that obeys to a normal distribution: Distance Sensors: $\mathcal{N}(distance, (5+\frac{distance}{10})^2)$ and Bearing Sensors: $\mathcal{N}(angleofview, 14.0625^\circ)$

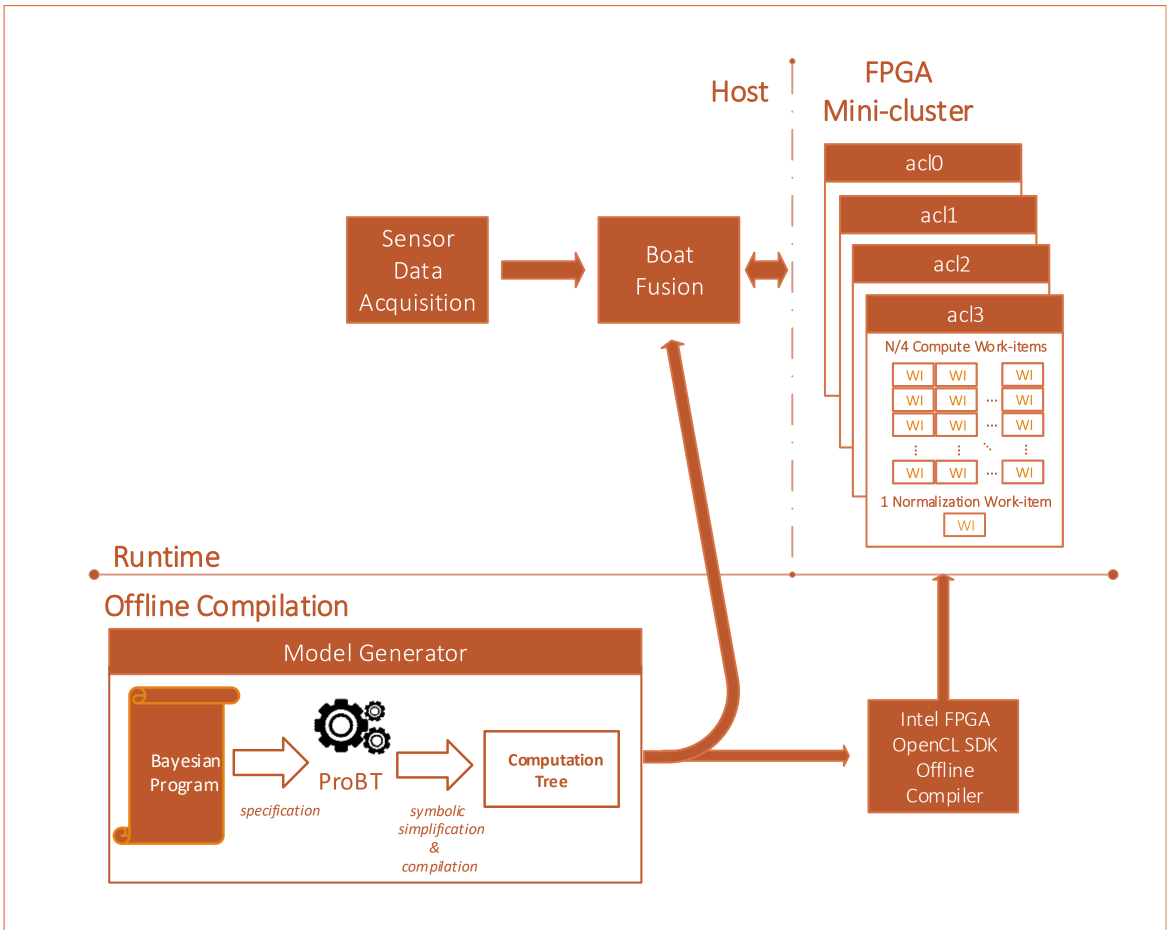


Fig. 3.7 System Architecture - From the Computation Tree generated by ProBT, the OpenCL host and kernels are compiled offline. During runtime, the OpenCL host program initializes the Compute Devices, loadsd the necessary memory buffers and enqueues the kernels for processing.

3.3.1 Model Generator

The Inference problem corresponds to computing the probability distribution over the X and Y coordinates, knowing the outputs of the six sensors. Therefore, considering:

- M_m = Position cell of index m
- E_k = Value of Sensor k
- $P(M_m)$ = Set of prior probabilities for each cell of index m
- $Posterior = \frac{Likelihood \times Prior}{Evidence} \Leftrightarrow$
- $P(model|data) = \frac{P(data|model) \times P(model)}{P(data)} \Leftrightarrow$

From Bayes Theorem, we gather that: $P(M|E) = \frac{P(E|M) \times P(M)}{P(E)}$. As the partition M_m is finite, we can use the Law of Total Probability to define $P(E) = \sum_m P(E|M_m) \times P(M_m)$. From the above, we can define our new Posterior Ratio as:

$$P(M|E) = \frac{P(E|M) \times P(M)}{\sum_m P(E|M_m) \times P(M_m)}$$

Considering that E is in fact a set of k multiple independent sensors E_k, we have:

$$y_m = \frac{x_m \times a_m}{z} \Leftrightarrow P(M|E) = \frac{\prod_k P(E_k|M) \times P(M)}{\sum_m \prod_k P(E_k|M_m) \times P(M_m)} \tag{3.1}$$

with:

- y_m : Posterior Ratio
- a_m : Prior Ratio
- x_m : Likelihood
- z : Normalization factor, scalar independent of of both m and k

We can formalize this problem as a Bayesian Program in the ProBT language. The ProBT API is then used to generate a simplified computation tree. In addition, the probability distribution across all variables is calculated offline and stored in Lookup Tables (LUTs).

$$
\textbf{Boat Locator}\begin{cases}
\textbf{Description}\begin{cases}
\textbf{Specification}\begin{cases}
\textbf{Variables}\\
D_1,D_2,D_3,B_1,B_2,B_3\\
\textbf{Decomposition}\\
P(D_1D_2D_3B_1B_2B_3|XY)=\\
=P(D_1|XY)P(D_2|XY)P(D_3|XY)P(B_1|XY)P(B_2|XY)P(B_3|XY)\\
\textbf{Parametric Form:}\\
P(D_1|XY)=\mathcal{N}(d_1,\,(5+\frac{d_1}{10})^2)\\
P(D_2|XY)=\mathcal{N}(d_2,\,(5+\frac{d_2}{10})^2)\\
P(D_3|XY)=\mathcal{N}(d_3,\,(5+\frac{d_3}{10})^2)\\
P(B_1|XY)=\mathcal{N}(b_1,\,14.0625)\\
P(B_2|XY)=\mathcal{N}(b_2,\,14.0625)\\
P(B_3|XY)=\mathcal{N}(b_3,\,14.0625)
\end{cases}\\
\textbf{Identification}\\
Alltablesprovidedbytheuser
\end{cases}\\
\textbf{Question:}\\
P(X\wedge Y|[D_1=d_1]\wedge[D_2=d_2]\\
\wedge[D_3=d_3]\wedge[B_1=b_1]\wedge[B_2=b_2]\wedge[B_3=b_3]
\end{cases}
\tag{3.2}
$$

3.3.2 Sensor Data Acquisition

The user provides the boat coordinates (x,y) as input to the OpenCL host program, this data is pre-processed by a routine and mathematically converted into sensor values. These new data is fed to the inference engine fro processing.

3.3.3 OpenCL Host - Boat Fusion

The OpenCL C++ host program effectively manages the inference computation. This program detects and initializes the available Compute Devices (FPGAs), creates the context and queues for each device, creates the buffers and loads them with the Lookup Tables and sensor data to be passed to the computing kernels. In addition, partitions the inference problem and distributes it through the queues of each Compute Device. It also passes the necessary control variables to each kernel. After the normalization kernel is finished, it stores and display the computed posterior probability distribution as a heatmap.

3.3.4 OpenCL Kernels

In order to optimize the inference performance, the computation was split in two separate kernels: The *Compute kernel* and the *Normalize kernel*.

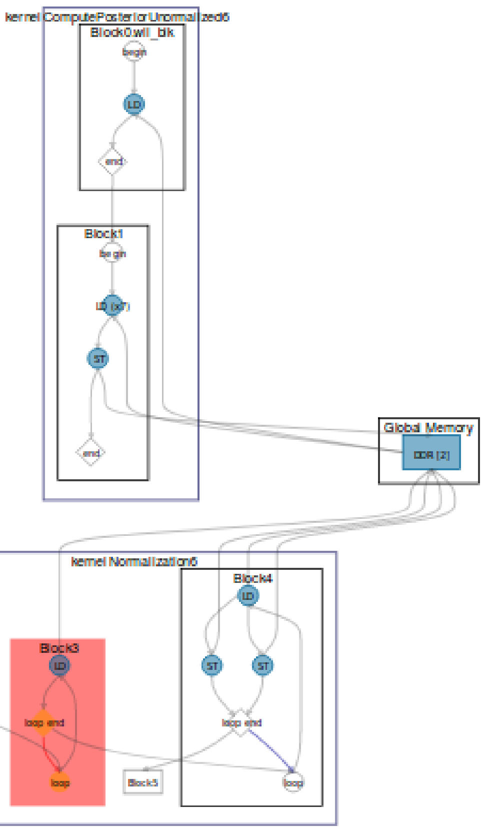


Fig. 3.8 Kernels Block Diagram.

The Compute kernel is massively paralelized, as OpenCL work-items. Each kernel instance performs the inference computation corresponding to one of the $n \in [0, 4096]$ grid cells. It loads the prior ratio from global memory, then it computes the indexes of the LUTs corresponding to the known sensor inputs and loads them from global memory. The computation is performed and the resulting ratio is stored to global memory.

After all instances of the compute kernel finish, The normalization kernel is launched. It normalizes the results and returns the posterior probability distribution.

As proposed in section 1.2, a Bayesian inference problem was successfully implemented on an Heterogeneous Computing Platform using OpenCL. Throughout development, attention was given to maintain the implementation as generic as possible. No vendor specific OpenCL optimizations were used, in order to allow seamless migration to platforms using different accelerator boards and FPGAs. In addition, the problem cardinality is controlled solely by pre-processor instructions and global variables, simplifying the scaling of the inference problem. The conversion from the ProBT computation tree to the OpenCL host was not fully automated, therefore, the implementation of different Bayesian Inference problems should require limited alterations to the developed code.

The work developed under this dissertation is generic enough to allow a seamless migration to GPUs. Scalability issues and energy efficiency degradation are expected due to the fixed hardware nature of the GPU. However, this limitations can be minimized by exploring the full potential of an Heterogenous Computing Platform with CPU, GPU and FPGA co-processors. The workloads can be partitioned during development in order to distribute the each computation to the fastest and more efficient Compute Device. A tradeoff is expected to arise between the computation gains and the memory speed bottlenecks between Compute Devices.

Chapter 4

Results and Analysis

The main objective of this work was to demonstrate the feasibility of an OpenCL implementation of Bayesian Inference on FPGAs. In order to achieve this goal, a generic Boat Localization problem was implemented. In this chapter the obtained results are presented and compared with data from previous work under the BAMBI project [11, 7, 8], .

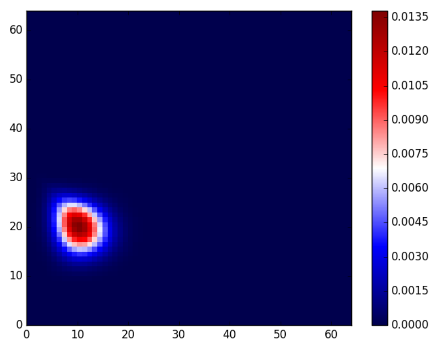
The problem presented is to determine the position of a boat in a discrete grid of 64x64 cells, therefore with a total number of possible positions $N = 4096$. In this demonstration, after the user manually introduces the boat coordinates, the readings of six noisy sensors are calculated and fed into the inference engine. Exact inference is performed and the resulting normalized posterior distribution is stored.

The resource usage, power consumption and processing speed are compared with a boat localization problem with the same size, implemented on the same FPGA mini-cluster using BAMBI´s BM1 and running for $10^4 iterations$.

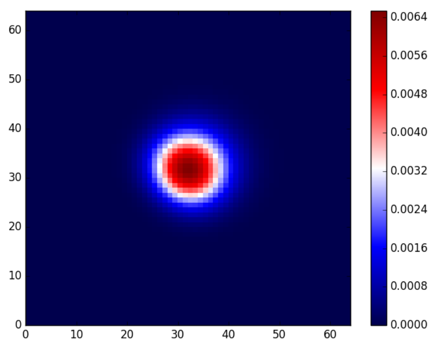
The computed probability distribution for four different boat positions are shown in 4.1, alongside the "real" boat position.

The results demonstrate that the Bayesian inference problem was successfully computed.

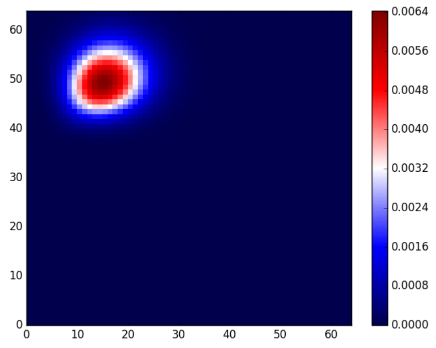
The inference problem was divided in two kernels. One computed the posterior ratio, while the other normalized the result. Due to its characteristics the first kernel was massively paralelized, performing the computations for all of the 4096 cells simultaneously. The normalization kernel, required a sequential access to memory, therefore became the bottle-



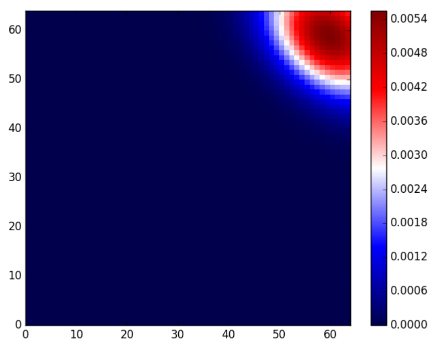
(a) Boat Position: (X,Y) = (10,20)



(b) Boat Position: (X,Y) = (32,32)



(c) Boat Position: (X,Y) = (15,50)



(d) Boat Position: (X,Y) = (60,60)

Fig. 4.1 Heatmap representing the probability distribution of the boat position in a discrete map of 64x64 cells. Distribution calculated through exact inference on the FPGA mini-cluster.

	Clock Frequency	Computation time	Sampling Frequency
OpenCL Exact Inference	249.6 MHz	1.28 ms	781 Hz
BAMBI BM1 Stochastic Exact Inference	50 MHz	0.2 ms	5 KHz
ProBT Exact Inference	N/A	1.127 s	0.89 Hz

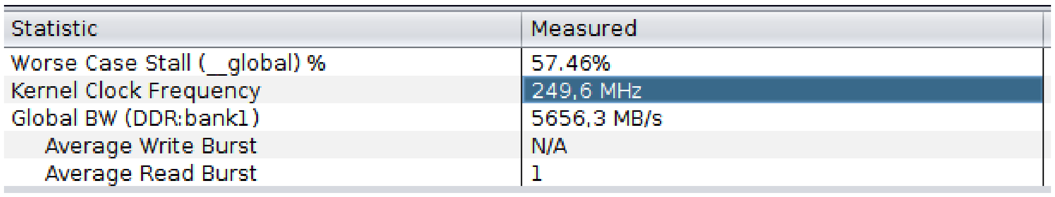
Table 4.1 Clock Frequency Performance

	Peak Power	Energy Consumption
OpenCL Exact Inference	0.0391 W	6.26 mJ
BAMBI BM1 Stochastic Exact Inference	1.15 W	2.30 mJ
ProBT Exact Inference	3.9 W	4.40 J

Table 4.2 Energy Consumption

neck of the system. The normalization kernel executed in 0,08ms, while the normalization took 1.2ms, resulting in a total of 1,28ms for the full inference computation. As expected, our implementation is slower than the original BAMBI BM1 (4.1), but significantly faster than a CPU implementation. Despite this fact, the frequency at we are able to process data (781 Hz), is fast enough to process the majority of similar robotic perception tasks in real time.

We were able to use a clock frequency five times higher than the BM1 implementation, at 249.6Hz, without stability issues. This stems from the smaller critical path in OpenCL implementation.



Statistic	Measured
Worse Case Stall (__global) %	57.46%
Kernel Clock Frequency	249,6 MHz
Global BW (DDR:bank1)	5656,3 MB/s
Average Write Burst	N/A
Average Read Burst	1

Fig. 4.2 Altera OpenCL SDK Profiling Report - Exact Bayesian Inference Kernels for Boat Example on a single device.

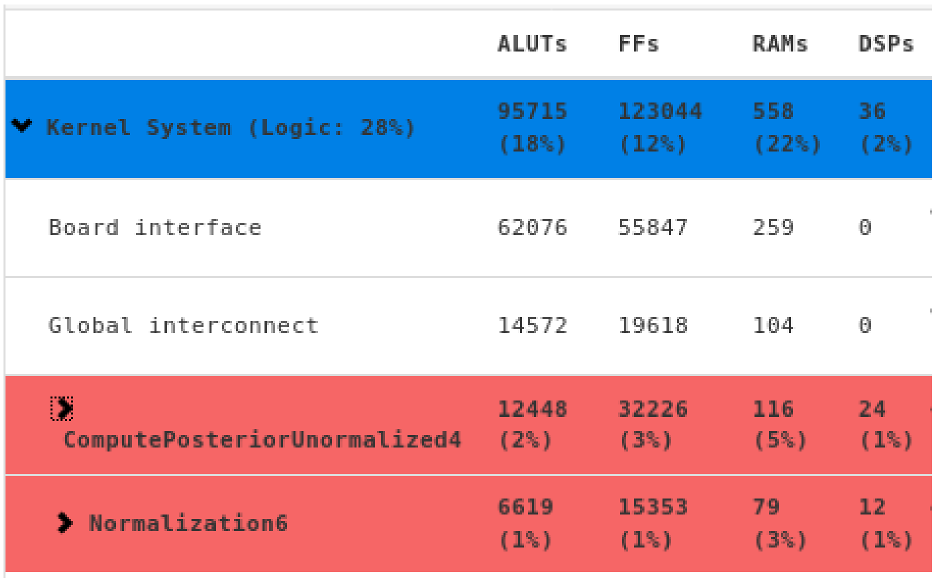
A smaller peak power usage4.2 was observed in comparison with BAMBI BM1, due to the reduced amount of hardware resources instantiated 4.3. However, as the OpenCL computation is is slower, the total energy usage is smaller in BAMBI´s implementation. In any case, the FPGA consumes significantly less energy than equivalent computation on a CPU.

The most unexpected result was the reduced hardware footprint. The hardware required for the complete inference, a fully paralelized compute kernel and normalization kernel

	Logic Elements
OpenCL Exact Inference	95715 (18%)
BAMBI BM1 Stochastic Exact Inference	178157 (68%)

Table 4.3 FPGA Resource Usage

occupy only 18% of a Stratix V FPGA. BAMBI´s BM1 on the other hand occupied 68% 4.3. In addition, the majority of the real-estate was used by OpenCL communication and control overhead, with the compute kernel only using 2% and the Normalization kernel 1%. This result greatly reduces BM1 scalability constraint. In our implementation, the boat´s grid size can be considerably increased, maintaining the same precision and similar processing speed without exceeding a single FPGA capacity.



	ALUTs	FFs	RAMs	DSPs
Kernel System (Logic: 28%)	95715 (18%)	123044 (12%)	558 (22%)	36 (2%)
Board interface	62076	55847	259	0
Global interconnect	14572	19618	104	0
ComputePosteriorUnormalized4	12448 (2%)	32226 (3%)	116 (5%)	24 (1%)
Normalization6	6619 (1%)	15353 (1%)	79 (3%)	12 (1%)

Fig. 4.3 Stratix V Resource Utilization of Exact Bayesian Inference Kernels for Boat Example on single device.

Further tests demonstrated that in OpenCL the distribution of the processing across multiple compute devices (FPGA mini-cluster) is a trivial problem. It was implemented by partitioning the map space in four identical subsets and instantiating identical kernels in each FPGA. The results showed through this approach, it is possible to significantly scale the cardinality of the problem without exhausting the FPGAs mini-cluster capacity.

Due to the computation speed achieved (781Hz), the possibility of time multiplexing the inference problem arises. For real world applications the refresh rate can be decreased, and therefore greatly increasing the cardinality of admissible inference. The combined use of time multiplexing and multiple Compute Devices reduces the tractability limitations of Bayesian Inference.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Throughout this work, an exact Bayesian inference task was implemented on an heterogeneous computing platform (2CPUs and 4 FPGAs) using OpenCL.

- Feasibility of OpenCL implementation of Bayesian Inference on FPGA mini-cluster demonstrated;
- Speed gains of an order of magnitude over exact inference implemented on CPU;
- Increased energy efficiency over exact inference implemented on CPU;
- Slower computational speed and increased energy consumption compared with exact inference on the same FPGA mini-cluster using Bambi's Bayesian Machine 1;
- Significantly higher FPGA resource usage compared with exact inference on the same FPGA mini-cluster using Bambi's Bayesian Machine 1 due to OpenCL overhead;
- Faster development cycle than using BAMBI toolchain and Hardware Description Languages (HDL);
- Less specialized development work, accessible to majority of programmers after some initial training. Specialized hardware engineers not required;
- Flexibility: the optimizations applied to the program are vendor independent, which should allow migration to different FPGAs and even GPUs with limited adaptations;
- OpenCL framework support by vendors still very limited;

- Several critical bugs encountered on the Gidel's ProceV Board Support Package (Drivers and peripherals information for OpenCL compiler), without resolution to this date;
- Various limitations discovered on the Intel FPGA OpenCL SDK due to compatibility issues with the older hardware in use. Most of the limitations are known bugs, that have been corrected in more recent versions of the software (but incompatible with Gidel ProceV boards);
- Intel FPGA OpenCl SDK is a resource intensive software during high level synthesis of the FPGA bitstream. The compilation of a kernel that occupies around 50% of Stratix V lasts in excess of 12 hours and is able to use all of the 64GB of available RAM.

This work has shown that even if the architectures pursued in BAMBI provide faster and more efficient circuits, the solution with OpenCL targeting FPGAs provides a very interesting intermediate solution, easier to deploy and potentially more scalable.

5.2 Future Work

The work produced during this thesis calls for further expansion and testing, specifically, we propose:

- Addition of a GPU to the system, in order to fully explore OpenCL load balancing capabilities, effectively dividing the workloads across Compute Devices according to its characteristics;
- Generalize and automate the toolchain in order to accept a broader range of Bayesian Inference problems;
- Integrate the toolchain with ROS (Robotic Operating System) to allow testing in real world conditions;

References

[1] A. Munshi, “OpenCL 1.2 Specification,” *Version 1.2*, p. 380, 2012.

[2] J. Shen, J. Fang, H. Sips, and A. L. Varbanescu, “An application-centric evaluation of OpenCL on multi-core CPUs,” *Parallel Computing*, 2013.

[3] J. F. Ferreira and J. Miranda Dias, “Probabilistic Approaches to Robotic Perception,” *Springer*, vol. 91, pp. 1 – 259, 2014.

[4] International Technology Roadmap for Semiconductors (ITRS), “More Moore,” *Itrs*, pp. 1–52, 2015.

[5] D. Weller, F. Oboril, D. Lukarski, J. Becker, and M. Tahoori, “Energy Efficient Scientific Computing on FPGAs using OpenCL,” *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA ’17*, pp. 247–256, 2017.

[6] P. BAMBI, “BAMBI Project website .” https://www.bambi-fet.eu/, 2015. [Online; accessed 1-May-2018].

[7] P. BAMBI, “BAMBI D3.10 : Emulation of a probabilistic computer on current reconfigurable logic,” tech. rep., 2017.

[8] M. G. D. Mira, “Using an FPGA Mini-Cluster to Implement Bayesian Application-Specific Integrated Circuits for Robotic Applications,” 2017.

[9] H. Fernandes, M. A. Aslam, J. Lobo, J. F. Ferreira, and J. Dias, “Bayesian inference implemented on FPGA with stochastic bitstreams for an autonomous robot,” *FPL 2016 - 26th International Conference on Field-Programmable Logic and Applications*, 2016.

[10] K. Hill, S. Craciun, A. George, and H. Lam, “Comparative analysis of OpenCL vs. HDL with image-processing kernels on Stratix-V FPGA,” in *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, vol. 2015-Septe, 2015.

[11] A. Coninx, P. Bessière, E. Mazer, J. Droulez, R. Laurent, M. A. Aslam, and J. Lobo, “Bayesian sensor fusion with fast and low power stochastic circuits,” *2016 IEEE International Conference on Rebooting Computing, ICRC 2016 - Conference Proceedings*, 2016.

[12] H. R. Zohouri, N. Maruyamay, A. Smith, M. Matsuda, and S. Matsuoka, "Evaluating and Optimizing OpenCL Kernels for High Performance Computing with FPGAs," *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, pp. 409–420, 2017.

[13] C. Farber, R. Schwemmer, J. Machen, and N. Neufeld, "Particle identification on a FPGA accelerated compute platform for the LHCb Upgrade," 2017.

[14] C. Färber, "Experience with Hybrid Intel Xeon FPGA System." 2016.

[15] Z. Wang, S. Zhang, B. He, and W. Zhang, "Melia: A MapReduce Framework on FPGAs, OpenCL-based FPGAs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9219, no. c, pp. 1–14, 2016.

[16] J. D. Alves, J. F. Ferreira, J. Lobo, and J. Dias, "Brief Survey on Computational Solutions for Bayesian Inference," *Workshop on Unconventional computing for Bayesian inference at IROS2015*, no. October, 2015.

[17] V. M. Morales, P.-H. Horrein, A. Baghdadi, E. Hochapfel, and S. Vaton, "Energy-efficient FPGA implementation for binomial option pricing using OpenCL," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*, pp. 1–6, 2014.

[18] J. F. Ferreira, P. Lanillos, and J. Dias, "Fast Exact Bayesian Inference for High-Dimensional Models," 2015.

[19] P. Bessiere, E. Mazer, J.-M. Ahuactzin, and K. Mekhnacha, "Bayesian Programming," *CRC Press*, no. 1, p. 380, 2014.

[20] M. Faix, J. Lobo, R. Laurent, D. Vaufreydaz, and E. Mazer, "Stochastic Bayesian Computation for Autonomous Robot Sensorimotor Systems," *Proceedings of the IROS2015 workshop on Unconventional computing for Bayesian inference*, pp. 27–32, 2015.

[21] Altera, "A New FPGA Architecture and Leading-Edge FinFET Process Technology Promise to Meet Next-Generation System Requirements," pp. 1–28, 2015.

[22] NVIDIA, "Tesla K8 GPU Active Accelerator," no. September, 2014.

[23] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," *FPL 2016 - 26th International Conference on Field-Programmable Logic and Applications*, no. c, 2016.

[24] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, S. Heil, J. Y. Kim, D. Lo, M. Papamichael, T. Massengill, D. Chiou, and D. Burger, "A Cloud-Scale Acceleration Architecture," *IEEE Micro*, 2017.

[25] C. Plessl, "Cray Commissioned to Deliver FPGA-Accelerated Supercomputer to Paderborn University ." https://pc2.uni-paderborn.de/about-pc2/announcements/news-events/article/news/cray-commissioned-to-deliver-fpga-accelerated-supercomputer-to-paderborn-university/, 2018. [Online; accessed 1-May-2018].

[26] I. FPGA, “Intel FPGA OpenCL SDK.” https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html, 2017. [Online; accessed 1-May-2018].

[27] Xilinx, “SDAccel - Xilinx OpenCL SDK.” https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html, 2017. [Online; accessed 1-May-2018].

[28] A. D. George, M. C. Herbordt, H. Lam, A. G. Lawande, J. Sheng, and C. Yang, “Novo-G#: Large-Scale Reconfigurable Computing with Direct and Programmable Interconnects,” 2016.