A DEDICATED SERVER DESIGN FOR PHYSICAL WEB APPLICATIONS

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by
Anes ABDENNEBI

July 2019
İZMİR

We approve the thesis of **Anes ABDENNEBI**

Examining Committee Members:

Assist. Prof. Dr. Semih Utku
Department of Computer Engineering, Dokuz Eylul University

Assist. Prof. Dr. Işıl Öz
Department of Computer Engineering, Izmir Institute of Technology

Assoc. Prof. Dr. Tolga Ayav
Department of Computer Engineering, Izmir Institute of Technology

16 July 2019

Assoc. Prof. Dr. Tolga Ayav
Supervisor, Department of Computer Engineering
İzmir Institute of Technology

Assoc. Prof. Dr. Tolga Ayav
Head of Department of
Computer Engineering

Prof. Dr. Aysun SOFUOĞLU
Dean of the Graduate School
of Engineering and Sciences

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Assoc. Prof. Dr. Tolga Ayav, The Chair of the Department of Computer Engineering, Faculty of Engineering at Izmir Institute of Technology. The door to Prof. Tolga's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this project to be my own work but steered me in the right the direction whenever he thought I needed it. Also, the Prof. endured my continuous prolonging of writing my final dissertation and guided me until I finished the whole work.

I must express my very profound gratitude to my dear parents and to my wife *Manel* for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without her. Thank you.

Finally, to all my friends and especially *Jawhar*, my thanks go to them also for supporting me during my work on the thesis.

ABSTRACT

A DEDICATED SERVER DESIGN FOR PHYSICAL WEB APPLICATIONS

With the huge impressive technological improvements the world is witnessing where giants like Facebook, Google, Apple, Microsoft and other technology companies are offering different services to millions of clients, services which don't take usually more than seconds to be within the users' devices besides the *Physical Web* applications that makes things interacts, having entities and can be reached based on the proximity context without omitting the incoming IoT infrastructure that would make 20.4 billion devices connected by 2020, the amount of data transferred, and services provided will be enormous and along with that, the big energy consumer standing behind providing clients with the needed data and services instantly, the web servers. Although it has a magnificent performance and responds to billions of queries and requests, however, there is still a crucial point which must be highlighted, the remarkable amounts of energy consumption by these servers. Therefore, this work is proposing a new approach in order to reduce the energy consumption in such a scenario where the 18-core energy efficient computer *Parallella* board will be used in order to create an energy efficient server that can offer many services triggered by various devices or any ordinary web requests across any environment and to prove also that using a cluster of *Parallella* supercomputers may perform as other similar servers dealing with web content (e.g. Raspberry Pi server). We will show how would these boards work under low energy feeding where users can access a web content hosted on a *Parallella* cluster. The source codes of the project are available on GitHub.[1]

[1] https://github.com/RoronoaZ/PARALLELLA.git

ÖZET

FİZİKSEL WEB UYGULAMALARINA ADANMIŞ SUNUCU TASARIMI

Facebook, Google, Apple, Microsoft ve diğer teknoloji şirketleri devlerin milyonlarca müşteriye farklı hizmetler sunmaktadır. Kullanıcıların içinde olmaları genellikle birkaç saniyeden uzun sürmeyen servislere tanıklık ediyor. Her şeyi etkileşime sokan, varlıklara sahip olan ve yakınlık bağlamına göre ulaşılabilen Fiziksel Web uygulamalarının yanı sıra, 2020'de bağlanan 20.4 milyar cihazı yapacak olan IoT altyapısını kaldırmadan, aktarılan veri miktarı ve sağlanan hizmetler çok büyük olacaktır. ve bununla birlikte, müşterilere anında ihtiyaç duyulan veri ve hizmetleri, web sunucularını sağlamanın arkasında duran büyük bir enerji tüketicisidir. Her ne kadar muhteşem bir performans gösterse ve milyarlarca soruyu ve *HTTP* talebi yanıtlıyor olsa da, bu sunucular tarafından dikkat çeken önemli miktarda enerji tüketimi vurgulanmalıdır. Bu nedenle, bu çalışma, enerji tüketimini azaltmak için 18 çekirdekli enerji verimli bilgisayar Parallella kartının çeşitli cihazlar tarafından tetiklenen birçok hizmeti sunabilecek enerji verimli bir sunucu oluşturmak için kullanılacağı bir senaryoda enerji tüketimini azaltmak için yeni bir yaklaşım önermektedir. Ayrıca, Parallella kümesini kullanılarak web içeriği ile ilgili diğer benzer web sunucuları gibi çalışabileceğini de kanıtlanmaktadır. Bu panoların, kullanıcıların Parallella kümesinde barındırılan bir web içeriğine erişebilecekleri düşük enerji beslemesi altında nasıl çalışacağını gösterilecektir. Projenin kaynak kodları GitHub'ta mevcuttur[2].

[2] https://github.com/RoronoaZ/PARALLELLA.git

TABLE OF CONTENTS

LIST OF FIGURES ix

LIST OF TABLES xi

CHAPTER 1. INTRODUCTION 1

1.1 Problem Statement 1

1.2 Objectives 2

1.3 Thesis Organization 2

CHAPTER 2. LITERATURE REVIEW 4

2.1 Concepts 4

2.2 Parallella 5

2.2.1 Overview 6

2.2.2 Architecture 7

2.3 Physical Web 9

2.3.1 Introduction 9

2.3.2 How it works 9

2.3.3 Beacons 9

2.3.4 Beacons protocols 10

2.3.5 What can be done with beacons? 10

2.3.6 Summary 10

CHAPTER 3. CLUSTER SETUP 12

3.1 Prerequisites 12

3.1.1 Hardware 12

3.1.2 Software 12

3.1.3 Benchmarking 14

3.2 Setup process 14

3.2.1 Components 14
3.2.2 Cluster software 15
3.2.3 Cluster setup 16
3.3 Issues 19
3.3.1 Operating System compatibility and Support 19
3.3.2 SD Card space allocation 19
3.3.3 Parallella's IP address possible changing 20
3.3.4 Epiphany SDK Build issues 20
3.3.5 Missing Hydra library for mpi 20
3.4 Summary 20

CHAPTER 4. FPGA AS A WEBSERVER 22
4.1 Architecture 22
4.2 General Architecture of the Embedded Server 23
4.2.1 A Potential drawback 23

CHAPTER 5. EXPERIMENTAL ANALYSIS 25
5.1 The CPU architecture 25
5.2 Scenario Depiction 26
5.3 Parallel Programming 26
5.3.1 Parallel Computing 26
5.3.2 Parallel Programming 27
5.3.3 MPI (Message-Passing Interface) 27
5.3.4 OpenMP 27
5.3.5 OpenMPI 28
5.4 The initial scenario 28
5.5 The second scenario 29
5.6 The Web Application (for benchmarking) 30
5.6.1 Sockets 30
5.6.2 Websocket 31
5.7 Siege (Load Testing) 32

CHAPTER 6. INTEGRATION IN THE PHYSICAL WEB ENVIRONMENT.....34
6.1 Infrastructure for the Parallella Cluster 34
6.2 The Physical Web Model 35
6.3 Self-Order Model 36
6.4 A Secondary Web Server 36

CHAPTER 7. POWER CONSUMPTION MEASUREMENTS 38
7.1 Power equation 38
7.2 Comparing with similar boards 42
7.2.1 Results discussion 44
7.3 Summary 44

CHAPTER 8. CONCLUSION.....46

REFERENCES.....48

APPENDICES
APPENDIX A. SOLVING THE SPACE ALLOCATION PROBLEM 50
APPENDIX B. EPIPHANY SDK BUILDING UNSOLVED PROBLEM 54
APPENDIX C. THE CLUSTER WORK DEMONSTRATION 56

LIST OF FIGURES

Figure		Page
1	Parallella Features & Specs Summary	6
2	The Parallella board illustration	7
3	Mesh Network for Epiphany	7
4	Parallella high level architecture (Source:[16])	8
5	A simplified architectural view of Parallella (Source:[16])	8
6	An Example of Choosing Ubuntu Dist. For Parallella z7010 Board (Source:[17])	13
7	SSH Configuration - IP address entering	17
8	Enabling X11 Forwarding	17
9	The Cluster Architecture after adding FPGA as an embedded server	23
10	The first scenario for a cluster with 54 cores in parallel	29
11	The scenario of sending http requests to the cluster	32
12	The results after testing the cluster with 1352 hits	32
13	The results after testing the cluster with 38989 hits	33
14	The Parking Meter Model	35
15	The restaurant model	36
16	Parallella Cluster as a Secondary Server	37
17	The Power & Current (I) readings for sensitivity of 0.1	39
18	The Power & Current (I) readings for sensitivity of 0.066	40
19	Data charts for the first table results readings	41
20	Data charts for the second table results readings	42
21	Classifying the boards according to their current consumption	43
22	Classifying the boards according to their power consumption in Watt	44
23	The unallocated space inside one Parallella board	50
24	The new partition after unmounting the partition	51
25	Resizing the new partition to 1MB	51
26	Resizing the new partition to 1MB - 2	52
27	Expanding the size of the OS partition	52
28	The final result of the allocating process	53

29 The esdk building error 55
30 Building the epiphany libraries separately (failed) 55
31 Responding in parallel to the requests sent from the web page 56
32 The web page and the process of receiving the http requests by the cluster 57

LIST OF TABLES

Table **Page**

1 The number of successfully responded transactions for different number of concurrent users for sensitivity = 0.1 mA/V 40
2 The number of successfully responded transactions for different number of concurrent users for sensitivity = 0.066 mA/V 41
3 Comparison of different boards' power consumptions under several states 43

CHAPTER 1

INTRODUCTION

As current standard web servers are providing millions of people an enormous amount of instant accesses to static or dynamic web pages, web applications, social network sites, online games, online shopping websites and other various internet services, the rate of energy consumed by these servers is extremely high and in most cases, there are some situations where these servers are still running and consuming power without providing any services. Besides the rising of the IoT era where it is estimated that 20.8 billion devices will be connected according to *Gartner* Inc. and 30.7 billion connected devices by 2020 according to *IHS* Ltd.[7]. We can make an assumption that the increase in the number of devices connected to servers and clouds will require more services and therefore more power in order to satisfy the users' needs. Thus, we believe that the amount of wasted energy will be high too so as a result there is money spent in vain.

Cluster computing, the concept which came to the foreground recently is going to be the expected solution for the issue of wasting power without providing services, and its concept is based on connecting multi computers over a network in order to form a cluster we think it is capable of performing as normal standard servers using 18-core credit card sized computer *Parallella* 0 which needs only 5W to be powered and offers high computation capabilities.

1.1 Problem Statement

Starting from the point that we want to reduce the amount of wasted energy by normal servers, parallella offers a good opportunity to achieve efficiency in power consumption.

As parallella was introduced first in 2012 0 making a cluster of these boards is a challenge in the matter of limits of some resources and the bugs that come out in the process of developing such a server.

So, our main aim is to make a cluster of 4 parallella boards in order to form a web server dedicated for some physical web applications, besides, for testing reasons we are going

to run tests using *Siege* by sending huge amounts of *http* requests to our server and measure response time and power consumption, by doing this we believe that an equivalent performance to standard servers can be obtained plus achieving energy efficiency through our parallella cluster.

1.2 Objectives

Reaching the final aim of this project will go through many goals that must be done:

1- Creating an efficient parallella cluster and installing the appropriate operating system on it.
2- Overcome the different problems that we may face concerning IP addressing, compatibility of the operating system with boards, remote SSH connection, opening GUI window from remote.
3- Measure the cluster performance and compare it with standard servers statistics and other cluster made of Raspberry Pi boards.
4- Test the cluster with a scenario where we make the interaction between physical web application and our server in order to test its efficiency in a matter of response time and energy consumption.
5- Measure the power consumption using another board which is an Arduino UNO.

1.3 Thesis Organization

After Chapter 1 which is an introduction to the dissertation and its clear statement of the problem and the objectives, Chapter 2 will give an overview about all materials and literature needed in order to have a good and comprehensive view about this project. The manner of building the cluster from scratch and describing all the needed parts in this project is well illustrated and described in Chapter 3. The 4th Chapter is about a theoretical view of the FPGA in Parallella and the possibility of implementing it as a secondary web server inside the Parallella cluster. Chapter 5 is about the experimental analysis, this section goes over the potential scenarios of using this web server, the parallelism concept in which the server code was written to satisfy as many *http requests* as possible in parallel, also, the CPU architecture is described briefly in this chapter. The 6th Chapter has an overview of how this web server can be implemented in a physical web environment. Finally, Chapter 7 shows the fruit of this work where it contains the

different results acquired from testing the cluster across various platforms especially the power consumption readings which is one of the main objectives of this work. This chapter is followed by three appendices illustrating some installing instructions or critical problems.

CHAPTER 2

LITERATURE REVIEW

This project is pointing to the problem of consuming too much energy which led to unjustified waste of power, so introducing Parallella as our proposed solution for the stated issue will guide us to go through some previous work done by a cluster of Raspberry Pi boards where we have discovered many common points in the process of building up our own Parallella cluster and therefore we can even compare our experimental results with those of Raspberry Pi alongside with standard servers.

2.1 Concepts

- **Epiphany architecture**

It is a distributed shared memory architecture which is composed of an array of RISC processors. These processors can communicate with each other thanks to the mesh *network-on-chip* connecting them. A node inside each one of these arrays is a processor itself capable of running an operating system. It basically contains:

  - A superscalar RISC CPU that executes 2 floating point operations plus a local memory in each node.
  - Multicore communication infrastructure in each node (it includes a network interface).

- **FPGA**

Field Programmable Gate Array, is a semiconducting device that can be reprogrammed after the manufacturing in order to fit a desired functionality by the user. It is composed of many configurable logic blocks. It can be used in many fields like HPC (High-Performance Computing), in Data Centers, industrial and medical applications, video and image processing, wired and wireless communications. etc.

- **Mesh network**

The concept of mesh is used here only to demonstrate the topology of the epiphany cores and how they are connected to each other, they are connected like each core is connected to its neighbors with a number of links bounded by 2<=N° links<=4. This topology helps in reducing the overhead of communications between epiphany cores which improves the -at least theoretically- the parallel running time of programs.

- **Http server**

It is a machine that serves several clients with data hosted on it. It is also called an *http server* since it uses the *HTTP* (*HyperText Transfer Protocol*) in handling the web requests. The *Parallella* cluster web server is going to play the role of a web server answering queries from single or different sources which try to reach a standard web page (in the beginning) stored in the parallella cluster. It is going to be proven that the built web server can handle a considerable amount of simultaneous *http requests* and be at least a secondary web server in some cases which may reduce the overload on the main standard servers at peak times.

2.2 Parallella

Parallella was launched by Adapteva in a Kickstarter project in 2012 0 and it is a high performance and power efficient computer in the size of a credit card with a total of 18 cores (16 epiphany cores + 2 cores of ARM A9) and an FPGA. It runs on Linux OS and its different distributions including Ubuntu. Parallella also has a Gigabit Ethernet port giving us good help in building our cluster where boards can communicate with each other in a fast way without any retarding. Also, it has 1 GB SDRAM and SD card reader (where we can put the card that hosts our OS). Some other ports like HDMI and USB are optional but it is not preferred to use these ports for purposes of reducing the total amount of used energy which is the main goal of our project.

2.2.1 Overview

The table below resumes the entire specifications of the parallella supercomputer 0 As it shows, some models are different from others, this is why it is important to make properly the choice of your board accordingly with your project's requirements. The following figure shows the board and its main components.

Model	P1600	P1601	P1602
Mnemonic	"Microserver"	"Desktop"	"Embedded"
Host Processor	Xilinx Zynq Dual-core ARM A9 XC7Z010		Xilinx Zynq Dual-core ARM A9 XC7Z020
Coprocessor	Epiphany 16-core CPU E16G301		
Memory	1 GB DDR3		
Ethernet	Gigabit Ethernet		
Boot Flash	128Mb QSPI Flash		
Power	5V DC		
Storage	Micro-SD		
USB	No	USB 2.0 Host Port	
HDMI	No	Micro HDMI	
GPIO Pins	0	24	48
eLink Connectors	0	2	2
FPGA Logic	28K Logic Cells 80 DSP Slices	28K Logic Cells 80 DSP Slices	80K Logic Cells 220 DSP slices
Weight	1.3 oz (36 grams)	1.4 oz (38 grams)	
Size	3.5" x 2.1" x 0.625" (90mmx55mmx18mm)		
SKU	P1600-DKxx	P1601-DKxx	P1602-DKxx
HTS Code (Schedule B)	8471.41.0150	8471.41.0150	8471.41.0150

Figure 1 Parallella Features & Specs Summary

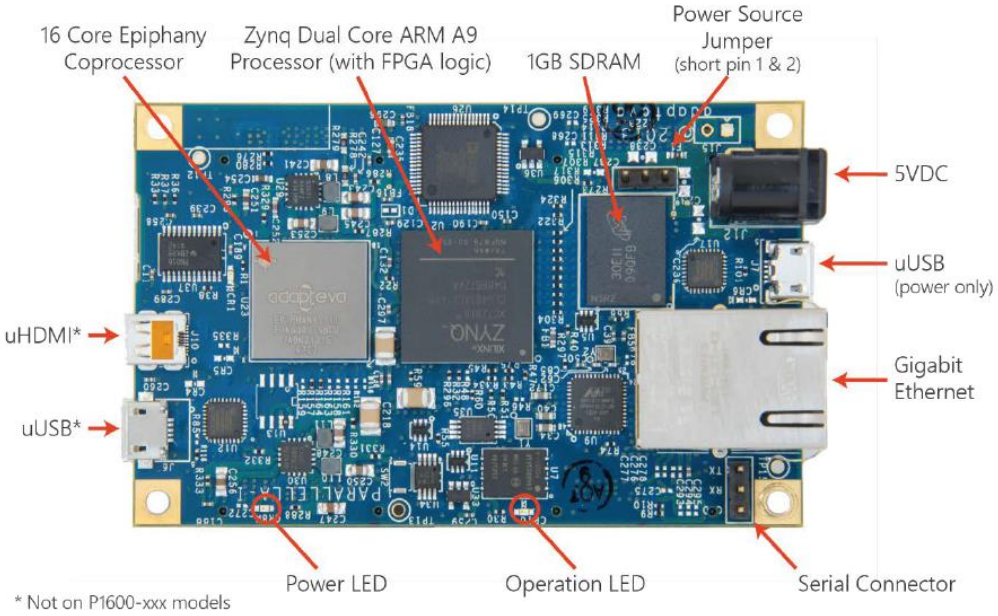


Figure 2 The Parallella board illustration

2.2.2 Architecture

This network connects the 16 nodes with each other allowing transactions (Read/Write) and communications between cores. For external communications, the mesh network permits that through I/O eLinks. One of the eLink IO bridges is connected to Xilinx FPGA which allows the epiphany cores and the ARM side to access shared memory area 0.

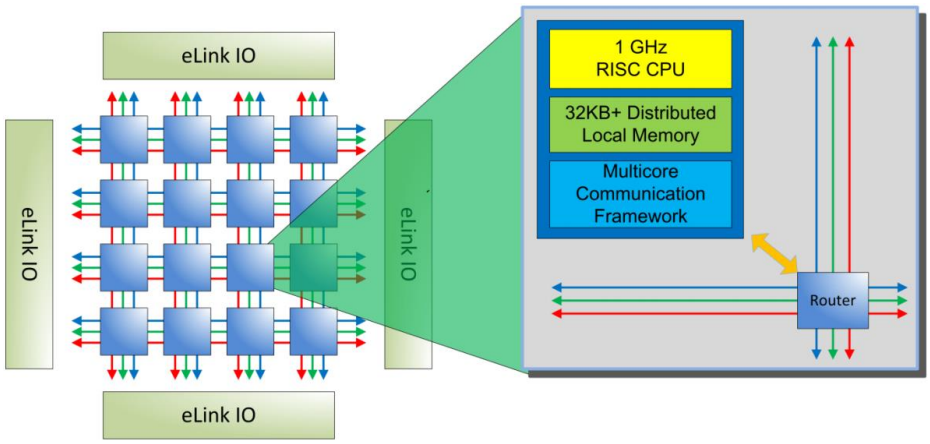


Figure 3 Mesh Network for Epiphany

A clearer diagram would explain parallella's high-level architecture easily and make it more explicit is shown in the following figure.

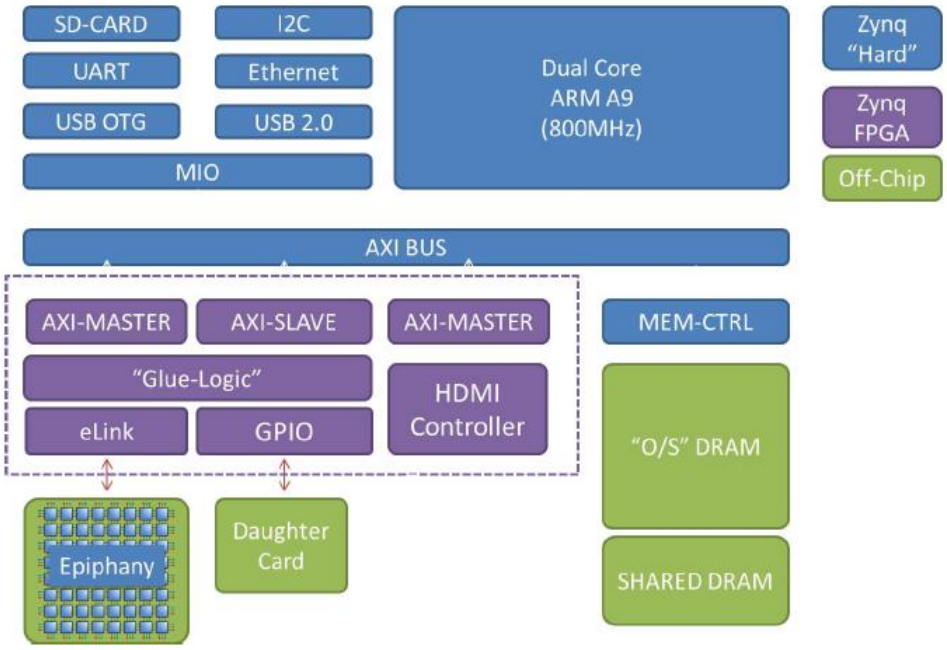


Figure 4 Parallella high level architecture (Source: [16])

And here is another diagram that shows how ARM-FPGA and the Epiphany 16 cores connect to each other in an abstract figure. We should drive your attention to a note that

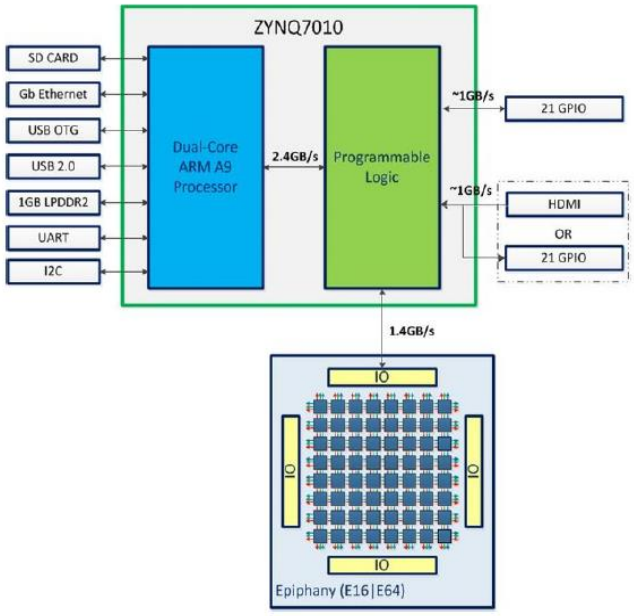


Figure 5 A simplified architectural view of Parallella (Source: [16])

despite the diagram is showing 64 E-cores but in our cluster, a 16 E-cores is used (Parallella Desktop P1601).

2.3 Physical Web

2.3.1 Introduction

The ability to access every object around you to get useful information without any identification or even need to Wi-Fi or mobile data connection (seamless connection) describes briefly the concept of the physical web.

At this point, someone could suggest that there are no other choices other than Bluetooth as a connection mean that can be established between our devices and other surrounding objects, objects that can be parking meter, voting wall, bus station, grocery items and many other objects that can be accessed through only URLs.

2.3.2 How it works

The physical web has the feature of showing a list of available proximity-based URL's list identifying nearby objects through beacons and these beacons can display their correspondent web links to some static or dynamic web pages, web apps or any other service. Another characteristic is that they run using Bluetooth Low Energy (BLE) protocol which doesn't require too much power to remain enabled.

2.3.3 Beacons

Beacons are small devices that work on low energy and hold a small amount of data also. And if you ask about the direction of the data flow then the answer would be in one direction where these beacons only transmit radio signals that can be detected by smartphones, tablets or any other smart devices, in other words, they don't receive any data, their task is to broadcast their data (probably URLs) to devices where some actions would be triggered afterward.

2.3.4 Beacons protocols

Even though turning on the Bluetooth of your device would detect any nearby beacons but there are some protocols that differentiate the way of how a beacon interacts with the device and what can it trigger.

- **iBeacon**

It is an Apple's company made protocol based on BLE (Bluetooth Low Energy). It is not that much different from other beacons but for Apple smart devices the scenario is a little bit changed because when they detect an iBeacon, it can launch an app in the iPhone or iPad unlike Android or other devices where a list of nearby beacons displays only some links to web pages or apps. We can summarize it as Apple flavor beacon [4].

- **Eddystone**

There are also other beacons that support *Eddystone*[3] protocol specification which can broadcast URLs.

2.3.5 What can be done with beacons?

Though beacons are tiny, and they do not perform any interaction with external smartphones or tablets, there is a lot to do with these data distributing devices:

- Positioning and navigation [5]
- Trigger actions for payments (not included within beacon)
- Tracking people or items.
- Making and order in a restaurant through a trigger to a web app page.

2.3.6 Summary

In this section, we took a brief look at our board parallella's architecture and specifications that we will use in the project, the mesh network which allows the

[3] Eddystone: is a protocol specification that defines a Bluetooth low energy (BLE) message format for proximity beacon messages [5].

communications internally between cores and externally between cores and ARM A9 & FPGA through the eLink I/O bridges.

Several software and applications will be used in this study, for building the cluster, configuring the beacons, benchmarking Parallella-based server. Therefore, we stated an overview over them before going deeper in details.

Due to the relationship that we are planning to establish between the parallella cluster and physical web applications, we introduced these tiny devices called beacons which through it, millions of devices can have access to numerous services and apps around the world. iBeacon & Eddystone are protocols defined by two different companies, their concept of working is the same, only some mechanisms of working may differ.

CHAPTER 3

CLUSTER'S SETUP

This chapter will discuss the procedures of getting the cluster built alongside with the required software for each step and if there are any other additional devices needed. The important part of this chapter -from our point of view- is the "**Issues**" section where many problems faced us building the cluster. Eventually, we'll conclude the chapter with an abridged summary.

3.1 Prerequisites

We can organize this section into two main parts; Hardware & Software.

3.1.1 Hardware

The parts needed for the setup can be listed as follows:

- A parallella board: 16 or 64 cores based on your project's needs.
- Sd card memory & its reader: 16 GB or higher (it will be explained why in the issues part).
- A network switch.
- 2 Ethernet cables: one cable to link the PC or Laptop and the other one to link parallella with the switch in order to be under the same network.
- An Ethernet/Usb adapter: in case your laptop does not have ethernet entry.

3.1.2 Software

In this part, we will introduce briefly each used software we used during our development of the Parallella cluster.

- **Ubuntu 14.04**

In order to choose the right ubuntu distribution for our parallella boards, we have to pay attention to the board's edition so we won't be mistaken choosing the compatible ubuntu version with the board as it is shown in the example is shown in the **figure** below.

- **Advance IP Scanner**

It is useful to discover the IP address that parallella took after installing all the stuff and turning the board ON. It is also important to note that the IP addresses of the boards may change due to some network protocols of assigning different IP addresses after a period of time.

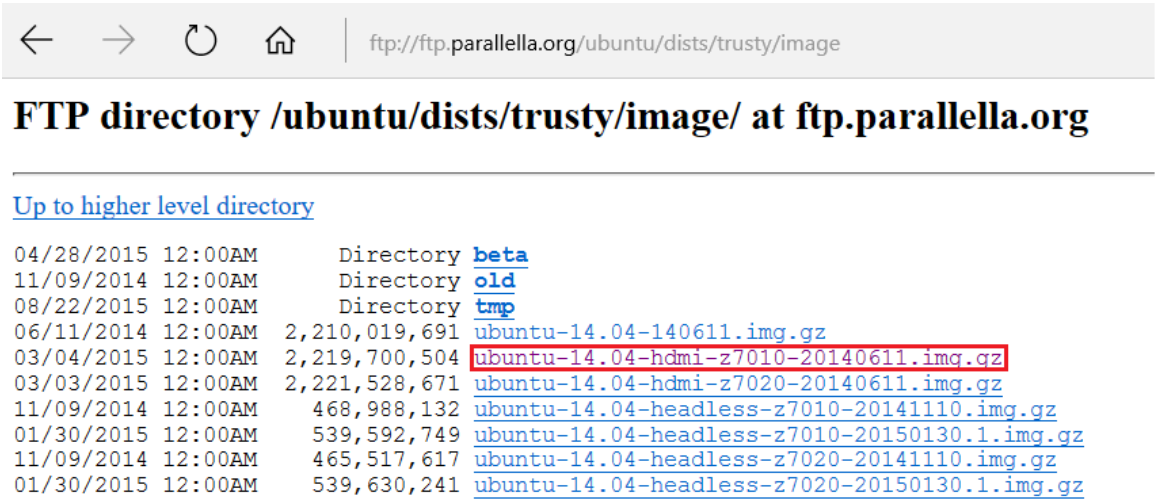


Figure 6 An Example of Choosing Ubuntu Dist. For Parallella z7010 Board (Source: [17])

- **Putty**

In order to have access to parallella through our machine, we have to launch an SSH connection using Putty software and in order to perform this step, we should have the board's IP first.

- **Xming**

Xming is an X.11 display server for Microsoft operating systems, we need it actually for displaying purposes when we need to launch Firefox navigator for testing the accessibility to an example *HTML* file stored in our server or sometimes to open the text editor *Gedit*. In general, if any GUI window is required to be opened during the process of our development of the parallella cluster Xming server must be in running mode.

- **Epiphany SDK**

As we have epiphany coprocessors so we need a development tool so as to program the epiphany cores therefore Adapteva[4] introduces ESDK (Epiphany SDK) based on the following development tools:

i. Optimized ANSI-C Compiler (GCC4.7)
ii. Multicore debugger (GDB7.3)
iii. Multicore communication and hardware utility libraries and other tools .

3.1.3 Benchmarking

- ***Jmeter***

It is an Apache application which is open source dedicated for load testing, testing the functional behavior and measure the performance of the machine under trial [2]. Jmeter can be used to simulate some loads which can be heavy or must be in order to push the server under test to its limits.

3.2 Setup process

First, it is a must to state clearly the components that were used during the assembly of the Parallella cluster accompanied by some figures supporting that.

3.2.1 Components

- ***Power supply***

it is preferable not to use other than a 3A rated 5V DC power supplier. Why it is 3A? 2 amps should do the job but in case of using the micro USB or HDMI connectors we go with 3 or 4A power suppliers but no more than this, for the purpose of preventing blowing a fuse. To be sure, just order it from Parallella's website 0.

- ***Ethernet RJ45 cables***

For our project, we have 4 Parallella boards so we need 4 Ethernet cables to link them with the ***Netgear*** switch.

[4] Adapteva is a semiconductor company that builds low power core microprocessor designs.

As a piece of advice for not getting confused when working on Parallella boards, try to make the cables short as much as you can, make its length suitable only for linking the boards because it won't look good with long cables getting mixed together.

- ***Two middle size fans***

It is highly recommended that you plug in two fans and in the middle of them, it is possible to put the cluster so one fan will cool down the boards from one side and from the other

side the second fan will send away the hot air coming from the working Parallella nodes.

- ***A simple Ethernet-to-USB adapter (optional)***

  When you try to link the cluster with your laptop with an ethernet cable and you don't have ethernet entry in your machine, you may need to use this adapter.

3.2.2 Cluster software

This section can be considered as the backbone of the project where nothing can be done without implementing it and assuring that it is well installed and configured.

- ***MPI (Message Passing Interface)***

It is a library specification for message passing that was developed to achieve high performance on parallel machines and on clusters. It addresses the message-passing parallel programming model where the data is moved from the address space of a process to another one through some cooperative operations on each process 0. MPI is going to be used in passing information and executing scripts in parallel ways and access all the cluster nodes from any node.

- ***MPICH***

It is a portable implementation of MPI with its different editions (MPI-1, MPI-2, MPI3) it supports different computation and communication platforms especially for our case (Multicore architectures) and it provides high-speed networks (10 Gigabit Ethernet) and other features which are out of our scope for now. One of its useful uses is using shared memory to pass messages faster between nodes. It has a free source license and its code is accessible 0.

- ***Apache2 (*Web server*)***

Apache HTTP (HyperText Transfer Protocol) server is an open source cross-platform web server and we are going to use it. It is open source so many developers can access the source code to fix, improve and collaborate in order to improve the Apache services, and it is cross-platform where it works in Windows, Linux or Mac OS.

3.2.3 Cluster setup

- **Putting all parts together**

After linking the power supplier with Parallella boards correctly and put them in a way where they can be cooled by a fan installed on one of the sides of the cluster to provide efficient cooling, we can move on and turn on the cluster and give it birth.

- **Detect the boards IP addresses**

Using "***Advanced IP Scanner***" start a scan for IP addresses within a specific range in order to detect the boards IP's. Usually, the name of Parallella's manufacturer "***Adapteva***" will be displayed under the manufacturer tab, this way we could pick our boards IP's.

- **Configure the SSH connections**

Using ***Putty,*** we had to set up some simple configurations concerning the SSH connection by entering the correct Parallella's board IP address, selecting the SSH port number (port No **22**) and without forgetting enabling the X11 forwarding (in order to be able to run ***gedit***, ***firefox*** or any other needed GUI software) as shown in the figures below.

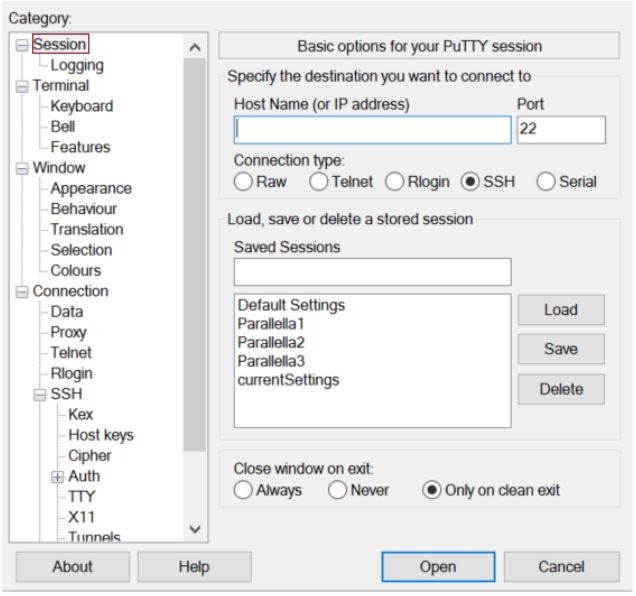


Figure 7 SSH Configuration - IP address entering

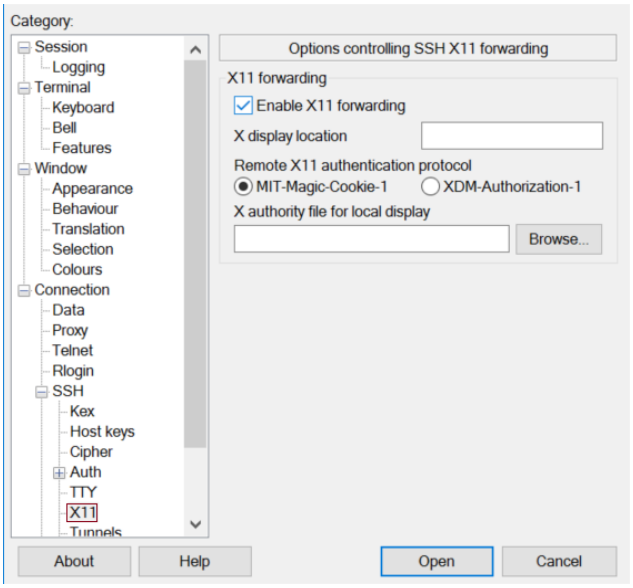


Figure 8 Enabling X11 Forwarding

- **Download & Install openmpi & MPICH2**

Considering openmpi, it comes pre-downloaded in the Parallella OS (in our case Ubuntu 14.04) so we just have to configure and make it (*./configure* && *./make* && *./sudo make install*) It may take a long time to accomplish this step, for us it took about 3 hours. Coming to MPICH2 we had to download it from its official website, once we did that, the remaining steps are only to install it.

- ## Cluster configurations

After finishing the previous steps, we only configured each board individually as a singular web server where each one of them has its own IP address with a page hosted within it and in order to make the whole cluster act as one web server we should proceed with the following procedures:

— Have another machine or board (it can be another Parallella, raspberry or Arduino) in order to use it as a **Load Balancer.**
— This Load Balancer has the job of generating Http requests and deliver them to the cluster.
— In our case, we used a laptop with Ubuntu 17.10 installed on a virtual machine as a load balancer because we had to write and modify a configuration script under Linux.
— By creating a load balancer, we added all Parallella boards IP's to the configuration script under the correspondent proxy tag.
— As an important remark; The **Load Balancer** needs to have two network interfaces for the aim of getting the http requests from the router in the first interface and send them to the cluster through the second one.
— The network that groups the load balancer with the cluster must be different from the network that hosts the interface linked to the router in order to run a simple graphical user interface for the load balancer (optional).

The next phase of configuring our cluster is testing it using a program called ***Siege*** which can be used to send a large number of http requests to test the cluster's response, durability and response time.

First, we have to install ***Siege*** by typing this simple command line:

```
siege -d10 -c10 -t1m http://Your_machine_ip/your_web_page.html
```

- The -c means how many concurrent users we want to add at a time.
- The -d specifies the delay between http requests.
- The -t tells us for how long the test should last (in the previous example the test will last for one 1 minute).

3.3 Issues

Since this project is new and went through unexplored topics, many issues and bugs faced us even in the slightest easy task. So we saw that making a whole section dedicated to describing these problems and giving its corresponding solution would be a great help for other colleagues or researchers who wish to work on a similar project or use ours as a starting point for further development.

3.3.1 Operating System compatibility and Support

As we stated before in the prerequisites, we should choose carefully the right distribution for the Parallella board we are using otherwise many problems would prevent us from making any step forward. Therefore, the chosen distribution should be identical to the Parallella model as shown in Figure 6.

Another point that we should highlight is the Ubuntu distribution support, as in our case that we faced where we first wrote Ubuntu 15.04 image on the sd card and ran it in Parallella and then in the process of installing Epiphany SDK prerequisites[5] an error appeared saying that it is unable to locate the libraries that we want to add. Thus, for those who chose ubuntu, it is preferable to avoid 15.04 distribution because it is no longer supported, even if you try to modify the "**sources.list**" file and replace the **archive** and **security** repositories with **old-releases**, it may work for some libraries but not for all.

3.3.2 SD Card space allocation

Another issue related to the storage capacity of the sd card, the chosen memory card's total space is 32GB and we couldn't figure out the issue until we started downloading the ESDK where it is required to have at least 9GB free space but unexpectedly the download failed due to insufficient space. It was so awkward and we tried many times with different sd cards but it was the same until we checked the sd card partitions through the **Disk management tool** in Windows and Gparted in Ubuntu. The problem was that after **Win32DiskImager** software wrote the Ubuntu image on the sd card it used the space needed for the OS (**about 7GB**) and left the rest (**about 22GB**) as Unallocated space which is neither formatted nor usable. The solution for this issue wasn't that easy to be

[5] It is referred in the section of building the SDK in Github repository for epiphany sdk through this link: https://github.com/adapteva/epiphany-sdk/wiki/Building-on-Linux

found and it took time to solve it especially that there is not many people working on Parallella and as a result, such a problem wouldn't be faced frequently.

Full documentation about how to solve this issue is presented in the appendix attached to the dissertation.

3.3.3 Parallella's IP address possible changing

It may not be a big deal but still can be an annoying one especially when you won't be able to login into Parallella, so pay attention to this little issue and try to fix its IP address as a permanent solution.

3.3.4 Epiphany SDK Build issues

During our cluster setup, we spent about one month trying to go through the issues we faced in the process of installing the ESDK and fix them one by one until we stopped completely when an error was declared during building the toolchain, a problem was pointing to *libtool* library. We tried to track the error and fix the missing, needed or unrecognized header files like (zlib.h) but there was no possibility to advance from that point, unfortunately.

In the appendix, there is a well-explained section describing this issue and the track of its roots.

3.3.5 Missing Hydra library for mpi

After installing openMPI and MPICH2 in each node (Master and worker nodes) and adding paths to build locations, we tried to execute a test script on both nodes (Master and Worker) as a test but the script didn't get executed on the worker node (we ran the code from the master node) and it gave the message shown in the figure below. So a download for the hydra library and installing it would fix the issue and make sure you are adding the correct paths to the right build locations.

3.4 Summary

It is substantial to note that some of the previously mentioned problems are not shown in any other project or reviewed by any group, some of the issues are figured out during the setup process and dealt with (the way of solving it is stated in the appendix). Therefore,

having a look at the appendix would be a quick fix for the problem without surfing the web looking for a solution and here is one of the advantages of making such project, to explore all possible issues and solve it.

Another point to make clear is to emphasize the important role that parallelism is playing in this project, the cluster is built by assembling 4 Parallella boards (2 of them are working unfortunately for some hardware defects) making use of all the cores in these boards will be beneficial for serving the maximum number of clients seeking for a web page hosted on the Parallella web server. Thus, some of openmp's main directives like *#pragma omp parallel*, are used in codes written in this project especially for the server side. So, a basic knowledge of parallel programming using OpenMP 0 is required to accomplish this work.

CHAPTER 4

FPGA AS A WEBSERVER

The Field Programmable Gate Arrays integrated into the Parallella boards has considerable importance as it can be a support for the web server running on the Parallellas by being itself an *Embedded webserver*. These kinds of web servers are widespread in various applications for many reasons where some can be reducing the overload on the main webservers, using it within printers, smart parking systems and many other contexts. Besides the expansion of IoT networks in almost every field like industry, communications, education, transportation. etc. has shaped the need for a new concept of the webserver. Therefore, in this section, an FPGA-based embedded web server architecture will be introduced and discussed although it was not possible to implement it in Parallella boards due to not hardware but software shortcomings. Moreover, it is important to note that the embedded web server architecture which will be proposed is completely theoretical and was not put into an experimental work, thus, the integrated FPGA's architecture and a possible scenario to configure it as a web server will be proposed.

4.1 Architecture

The used Parallella boards in this project have the System-on-Chip (SoC) Zynq Z7010 as a host processor and Epiphany 3 as a co-processor with an integrated FPGA with 28K Programmable Logic Cells. As Figure 5 shows, the Zync Z7010 hosts both the ARM A9 and the FPGA where the latter is connected to the ARM processor with a 2.4 GB/s link and to Epiphany with a 1.4 GB/s which shows that the FPGA is easily accessible by the ARM cores through AXI bus. The Programmable logic is linked to Epiphany and can access its cores through the eLinks.

The FPGA in parallella has the following features:

- LVCMOS, LVDS, and SSTL signaling with 1.2V to 3.3V IO
- Up to 125 programmable IO pins (Z-7020)

- Up to 85K programmable logics cells (Z-7020) 28K in (Z-7010)
- Up to 560 KB distributed RAM (Z-7020) 240KB in (Z-7010)
- Up to 220 DSP slice and (Z-7020) 80 DSP in (Z-7010) 0.

The Parallella Cluster works as a web server using the ARM A9 cores and if it were possible, there would have been an embedded server ran by the FPGA in the Zync 7010 which would be beneficial in increasing the number of served *http* clients but due to the software issue which made it arduous to modify the FPGA codes according to the desired needs. However, it is still substantial to propose a potential general architecture for the FPGA-based embedded server theoretically.

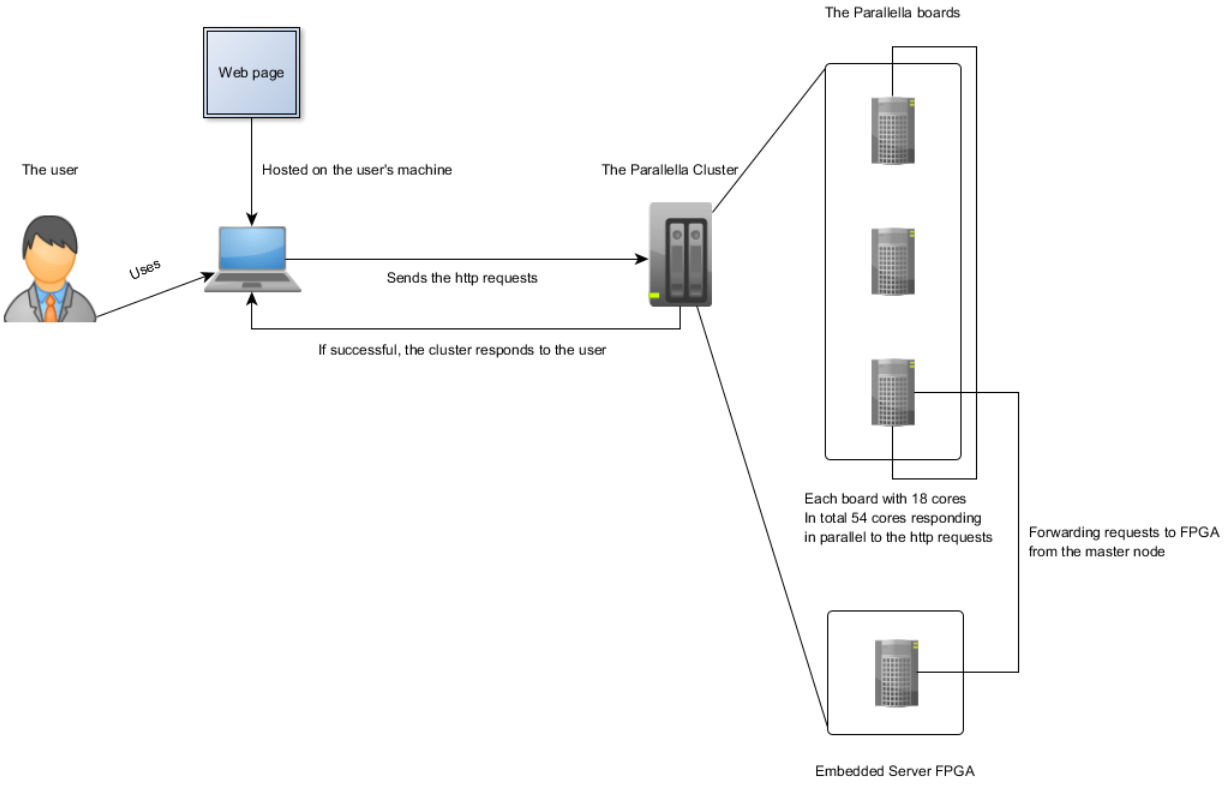


Figure 9 The Cluster Architecture after adding FPGA as an embedded server

4.2 General Architecture of the Embedded Server

4.2.1 A Potential drawback

In a scenario where multiple users (clients) are accessing the Parallella Cluster web server and as stated before in a previous section, the requests will go through the master node which runs a *message passing* command in order to run in parallel multiple Parallella boards as slave servers. The FPGA contribution might be as a server which also receives many *http* requests and treats them concurrently. One of many possible inconveniences

that this setup may cause is the communication overhead, the first hardware to receive the *http* queries is the ARM processor and in order to reach the FPGA there must be a communication link between them (presented as a 2.4GB/s link) but still, as a mere assumption, the existence of an *embedded web server* side by side to Parallella Cluster would increase the number of clients that this cluster can handle, and in the other side, it may also increase the latency of it.

CHAPTER 5

EXPERIMENTAL ANALYSIS

This section has two subsections where first, an experimental scenario will be set in order to test the cluster's functionality and its responding to normal http requests, also to figure out, how much of these requests this cluster can handle before crashing, so it can be called figuratively as *Stress Testing* for the parallella cluster.

The second subsection is about measuring the power consumption of the cluster during its functioning time and that would help to give conclusions about the power efficiency of this cluster which may or may not support the first claim about being a less energy consuming server than previous ones.

Based on the results extracted from both subsections, the full image will be clear and therefore, an epilogue can be drawn after comparing these results with other external experiments made on Raspberry-Pi web servers. Of course, some hardware and software unsolved issues have been faced during the installation of this cluster which would prevent it from benefitting from many privileges that would have probably helped in improving the response speed and time and also the amount of http requests that this cluster can handle, for example, the *Epiphany SDK* issue which wasn't solved even by its manufacturers, it stopped the progress of the project at a phase where using the parallella 16 cores was out of the question even by trying to use some of the pre-installed projects that used those cores but it was not possible since the parallella cluster required parallel programming to handle many requests at once but unfortunately, that was completely depending on the existence of the *Epiphany SDK*. However, some other methods were followed to compensate for this drawback which will be explained in the incoming subsections.

5.1 The CPU architecture

As a crucial point in this section, the architecture of the ARM A9 processor must be checked in order to know a-priori how to address the parallelism under this platform's

specifications. In a nutshell, the arm processor in Parallella has 1 socket containing two cores where each core has one thread, therefore, since there are 3 Parallella boards in the cluster it means that in total there are 3 cores and a 2x3 threads running in the whole cluster. In case that it was possible to use the other 16 Epiphany cores, there would be a total of 54 cores running 54 threads in parallel which would have been of great importance and increase the cluster's performance by accepting more *http requests.*

5.2 Scenario Depiction

Actually, this scenario was followed to test the cluster with a flux of web requests generated from any machine within the same network since the cluster was not assigned a public IP address because that would be too dangerous and can expose the parallella boards to any kind of a cyber attack where someone, for example, can simply make use of the boards computation capabilities.

So, at this stage, we assume that the cluster is set and ready to be used, correctly connected to the same local network that the tester machine is connecting to in order to avoid being in different local networks.

The machine can be a desktop computer, a laptop, a tablet or any smart device that can be attached to the network. For the next step, one can ask how to send requests to the cluster? Simply, a simple *web application* was created to simplify the interaction with this web server. The main and only job of this web interface is to send a specific amount (can be tuned according to the tester desire) of HTTP requests. The cluster will respond to these requests and by following this method, the stress testing can be performed in order to know until which upper bound of http requests can this cluster still function before crashing, that would help to set the limits of the parallella cluster under the installation restrictions and issues that were experienced previously.

5.3 Parallel Programming

5.3.1 Parallel Computing

It is a concept defined in [3] referring to a group of autonomous computers connecting to one another through communicating channels or lines. These computers may perform some computations, share data and generate outputs and these processes can run simultaneously.

Another definition of the parallel computing states that the parallel computing is the simultaneous use of many computing resources in order to solve a computational problem where this problem can be divided into sub-problems and distributed to many machines or processors to solve it in a parallel manner and that would save processing time and also organize the memory accesses.

5.3.2 Parallel Programming

It is using more than a processor to achieve a task where these tasks have some conditions like being **independent** and that means that the order of the processes is not important besides that it does not use the same data so, there is no risk of facing data race issues. That can be achieved by making each processor works on a sub-problem with a partition of the data that is distinct from any other data being processed by other processors or microcontrollers.

In this work, the first plan was to use all the cores of the parallella boards forming the cluster in a way that serves all the simultaneous HTTP web requests in a short time effectively. There are 3 parallella boards, each one with 18 cores, therefore, in total there are 54 cores that could be used in the cluster but unfortunately, because of the shortcoming of not being able to install the *Epiphany SDK*, it became hard or nearly impossible to benefit from such a computational resource. However, it is still bearable to use the *Message Passing Interface* to recompensate the absence of threads or any other parallel programming mechanism.

5.3.3 MPI (Message-Passing Interface)

It is a library specification for message passing that was developed to achieve high performance on parallel machines and on clusters. It addresses the message-passing parallel programming model where the data is moved from the address space of a process to another one through some cooperative operations on each process [8].

5.3.4 OpenMP

Is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, instruction set architectures and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows 0. There are many directives included within the *OpenMP*

library, sometimes under some systems, there must be a call for *omp.h* header file to include all its directives and make use of them. For example, in order to create a parallel region, there is a specific instruction for that which is *#pragma omp parallel* which means that inside this region there will be many threads operating on the code in parallel, but still the programmer should state the manner the threads are going to run the code (or part of the code) and how should they communicate between each other (either through a shared memory space or message passing interface).

In the case of Parallella Cluster, it is so simple to parallelize the code used for running the server since it contains an initiation of a server receiving web requests which can be considered as an unbreakable task (cannot be divided into chunks and assign each part to a different thread), however, it is possible to run the code on two threads working in parallel, why only 2 threads? It is due to the *CPU* architecture of Parallella which has only two cores and two threads.

5.3.5 OpenMPI

The *openMPI* directory which contains a version of *MPI* that needs only to be configured and built is found in the OS installed on Parallella. *openMPI* supports many compiler sets like C, C++, F77 & F90 (Fortran) and it is integrated with the operating system along with its compiler set 0.

5.4 The initial scenario

The initial plan before having the SDK problem was to make the Parallella cluster work with its all cores, which means with each Parallella board's 16 cores and that makes it in total 48 corse plus the ARM A9 processors and the FPGA's to at the end, the full potential of this cluster would have been 54 cores working in parallel. Thus, the scenario was to send *HTTP requests* to the cluster where the master node would be responsible for distributing the tasks to the other 2 boards and inside each Parallella, all the cores would be working in a *thread* that keeps working and handling the incoming requests in parallel with the other cores and moreover with the other board's cores.

This scenario would have given an extraordinary performance in handling the web requests simultaneously using the concept of parallel programming. However, since the abovementioned issue concerning the SDK is preventing this step from being achieved,

another direction in the project should be taken in order to at least introduce parallelism between Parallella board's which is achievable by depending on the *openMPI* directory installed on each Parallella and uses a version of the Message-Passing Interface, hence, it would be possible to attain at least an acceptable level of parallelism within the built cluster and therefore fulfilling the first goal of making this project and integrate it within a physical web infrastructure in further steps.

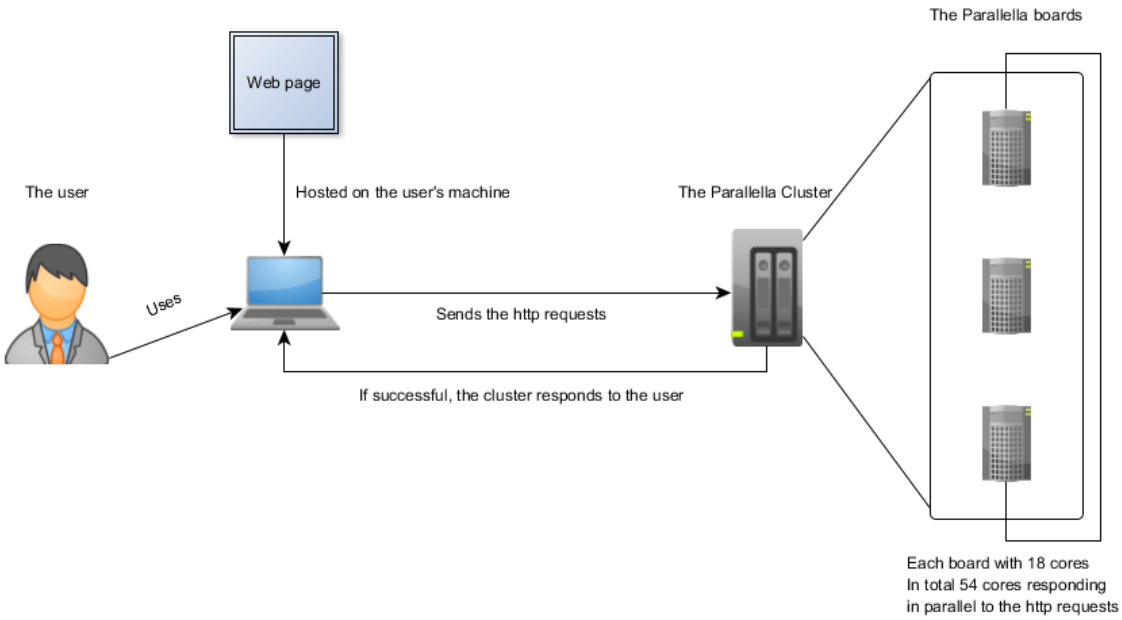


Figure 10 The first scenario for a cluster with 54 cores in parallel

The figure in below gives an illustration about what was explained earlier about the first planned scenario for this web server Parallella cluster.

5.5 The second scenario

In this situation when it is agreed to work only on the parallelism between the Parallella boards, the main focus will be on the *MPI* library to achieve message-passing between the nodes and benefit from this privilege to reach an acceptable degree of parallelism in answering the concurrent *HTTP* queries. Under these circumstances, the only change will be delivering the *HTTP* requests to the cluster nodes and only the A9 processors will be repsonsible for answering it, therefore, in total, there will be 3 processors working in parallel. The way these boards wil function together is by using *MPI* where the selected

master node of the cluster will run the server script (written in C) on all the nodes - including itself- waiting for incoming requests. The tester (it means the client but it is tester now since it is in the experimental phase) through the web application sends a

specific amount of *HTTP* requests to the cluster, the latter will receive these queries as same as the rest of the Parallella boards and after that, the responses can be noticed through the terminals of each board (for monitoring purposes). Moreover, each board will be running in parallel and

having two threads running on two cores which will push the server to its limits and try to achieve the maximum satisfied *http queries*. As an important remark, the *mpirun* command used to start up the server scripts on each node is responsible only for running the code on slave nodes without being accessed explicitly (i.e. without opening sessions on each node eand launch the scripts) and it is not responsible for any parallelism presented in this cluster.
In the appendix, under one section, some figures are illustrating the request-answer flux of the whole system (Web server -cluster- and the client -tester-).

5.6 The Web Application (for benchmarking)

It is simple for any machine located in the same local network with the Parallella cluster to test the access to a web page hosted on this cluster and make sure that it is working, but that won't help in testing the durability of the cluster, how many simultaneous requests it can handle and also it will be useful for the power consumption measurements where the final comparisons would be made based on these power readings when the cluster is performing at its maximum capability.

A simple web application is made using *Javascript* where the tester will be able to send multiple requests from any machine to the cluster and waits for the response which states that the requests were fulfilled successfully, meanwhile, the cluster terminal would be open to show when the HTTP requests come and getting responses. All of these steps will be written in details in the ***Appendix*** section which will help for any further projects on this kind of web server clusters based on Parallella boards.

5.6.1 Sockets

It is one endpoint of a two-way communication link between two programs might be hosted on the same or on different machines running on the network. A socket is bound

to a port number so that the TCP layer can identify the application that data is destined to be sent to 0.

To explain more the scenario where the sockets were used to test the Parallella cluster, some more captions need to made and clarified.

Actually, the web server runs on a machine (the parallella boards) and has a socket assigned to a port number waiting for a new connection from users. The users (clients in other words) in order to establish a successful connection with the server, they have to know the hostname and the port number of the server, also the clients need to binds to a local port number for the purpose of identifying themselves to the server. When the connection request is sent to the server and the credentials are correct, a new socket is created with the local port number in order to listen to the client and both the server and user can communicate through this socket.

This is briefly what happens between the parallella cluster and any potential client, with a slight difference where the cluster recieves the requests in a parallel way with all its nodes (the remaining parallellas) since the *openMPI* API was used.

5.6.2 Websocket

Since the sockets will be used in the communication between the clients and the server, there might be a slight difference in the nature of the socket that is going to be used according to the which environment is going to be used and also the programming language it is written with. In this work, the HTTP requests will be sent through a web application created by *Javascript* and benefitting from the privilege that the sockets are environment independent, therefore, a new socket will be created in the web page to send a connection request to the Parallella cluster which runs a server script and also has a socket written in C. With just a tiny variation where in the web page the sockets are defined in a structure called *Websocket*, which is a protocol of creating a fast two-way communication channel between the web browser and the server. Websocket has the privilege of updating the content of the website without waiting for the user 0.

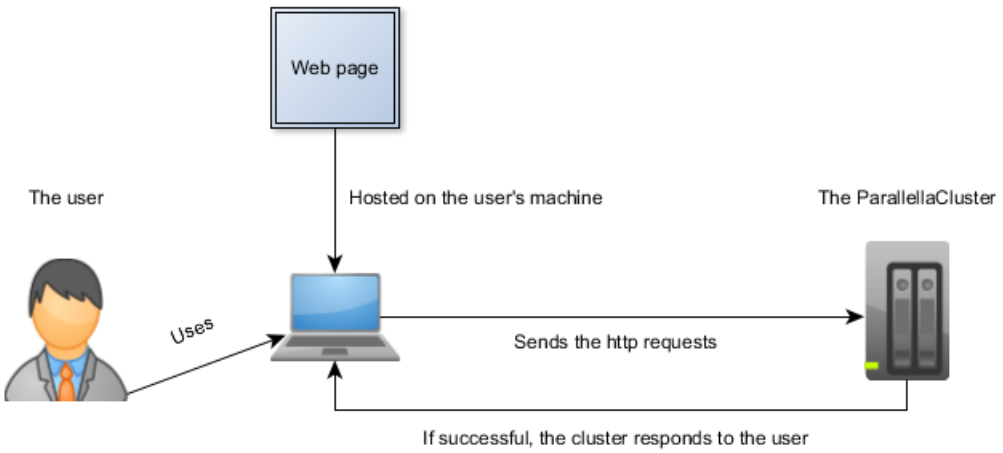


Figure 11 The scenario of sending http requests to the cluster

5.7 Siege (Load Testing)

Siege 0 is an http load testing tool for benchmarking built on GNU/Linux, it supports HTTP, HTTPS and also FTP. It would be a good idea to use it in this work since *Siege* permits its user to flood a server with a number of simulated clients and that would test the server's durability in handling different users http requests simultaneously, therefore, it will be experimentally possible to depend on Parallella Cluster in serveral implementations and scenarios.

The Parallella-Cluster was put to tests under many configurations like playing with the number of http hits and the number of concurrent simultaneous clients then observe the outcomes of these tests like the *Server Availability percentage, the Elapsed time, the*

```
File  Edit  View  Search  Terminal  Help
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer
[error] socket: read error Connection reset by peer sock.c:539: Connection reset by peer

Lifting the server siege...
Transactions:                   1352 hits
Availability:                 100.00 %
Elapsed time:                  59.66 secs
Data transferred:               0.50 MB
Response time:                  0.16 secs
Transaction rate:              22.66 trans/sec
Throughput:                     0.01 MB/sec
Concurrency:                    3.57
Successful transactions:        1352
Failed transactions:               0
Longest transaction:            3.33
Shortest transaction:           0.01
```

Figure 12 The results after testing the cluster with 1352 hits

Response time and *the Successful transactions.* Figure 12 shows the results after testing the server with ***1352 hits*** 0

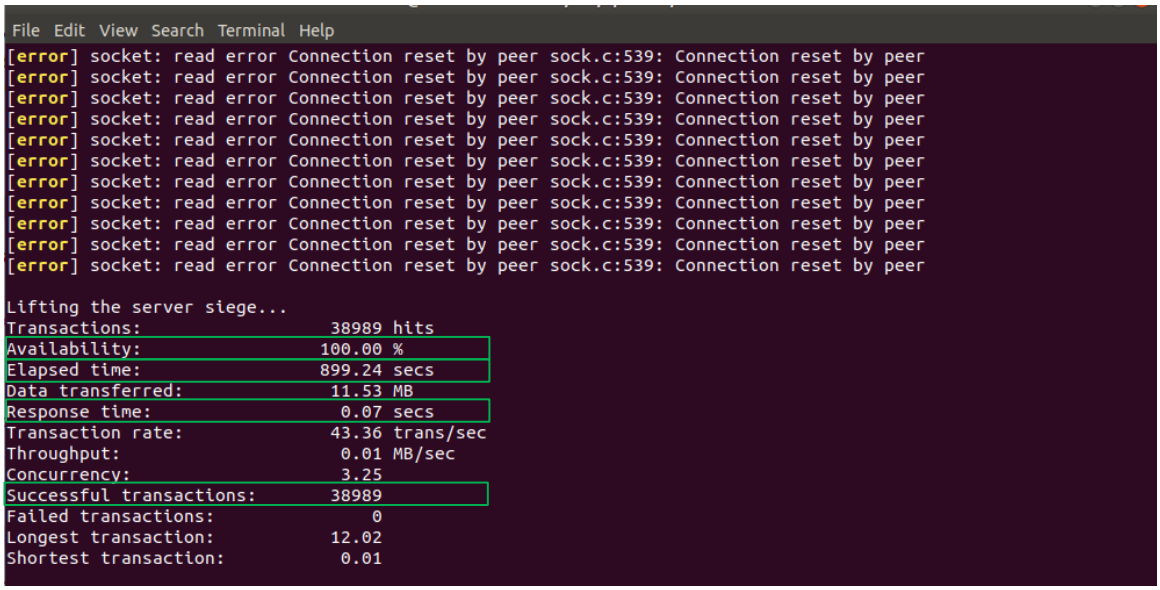


Figure 13 The results after testing the cluster with 38989 hits

Another test was made but this time with a much larger number of http hits, about 38989 hits, the results are presented in figure 13.

The Parallella-Cluster took about 1 minute to respond to all the 1352 http requests successfully in the test performed in figure 12 where it took about 15 minutes to answer the 38989 hits successfully. In both tests, 200 users were simulated to access a web page hosted on the Parallella Cluster.

It is important to note that someone can specify a txt file containing many URLs so that the server would access a list of URLs instead of one URL which is considered as more realistic test.

CHAPTER 6

INTEGRATION IN THE PHYSICAL WEB ENVIRONMENT

The main objective of this work is preparing a web server made of a cluster of microcontrollers (Parallella boards) that has the potential of handling multi *HTTP* requests concurrently where it can be implemented in a scenario integrated in many physical web infrastructures.

As it was stated in *subsection 2.3*, the Parallella cluster may replace the standard server in receiving the clients' requests when it has a public known address reachable by several applications having its address. The difference between using standard and Parallella-based servers is the power consumption, if the cluster built in this work is proved to be efficient in responding to the clients queries in short or acceptable latency rates then it can be said that the standard servers are consuming unjustified energy especially when it is dedicated for small segment of users or restricted environments, in this case, using the Parallella cluster would be reasonable and efficient plus, considered as an energy-saving solution.

6.1 Infrastructure for the Parallella Cluster

In this section, it will be clarified where this solution (the cluster) can be integrated, in which situations this web server can perform and give nearly the same performance as a normal server without any long latencies. But before that, it is important to state that the following scenarios won't be complicated or their uses aren't so crucial like smart factories, VANET based vehicule tracking models or any other use where it is required to have low latencies and any delays would cost money or even lives. Therefore, three examples of potential usages of the Parallella cluster will be introduced and explained with figures showing both the usability and feasibility of the cluster role in the proposed infrastructure. The first model is the physical web based model which is the essential mainstay of this work, the second one is a simple restaurant self-order system which can be considered also as a variant of a physical web application, and the last example is a

theoretical one which needs more studies or some field experiments and it is a secondary web server for supporting standard servers in situations such that the latter is facing some congestion problems where the servers reached their capacity limits, here the Parallella Cluster can handle few more requests by hosting the web page or application and preventing the service from being unavailable which is not desirable in most of cases.

6.2 The Physical Web Model

This model includes the beacons that were introduced in the *subsection 2.3*. Those beacons store some urls since it has very constrainted memory and power. Through these stored urls, any user near these beacons after establishing a bluetooth connection can obtain the url stored in the beacon in order to reach a specific web service or application. This model is about implementing the cluster as a *Cloud Server* responsible

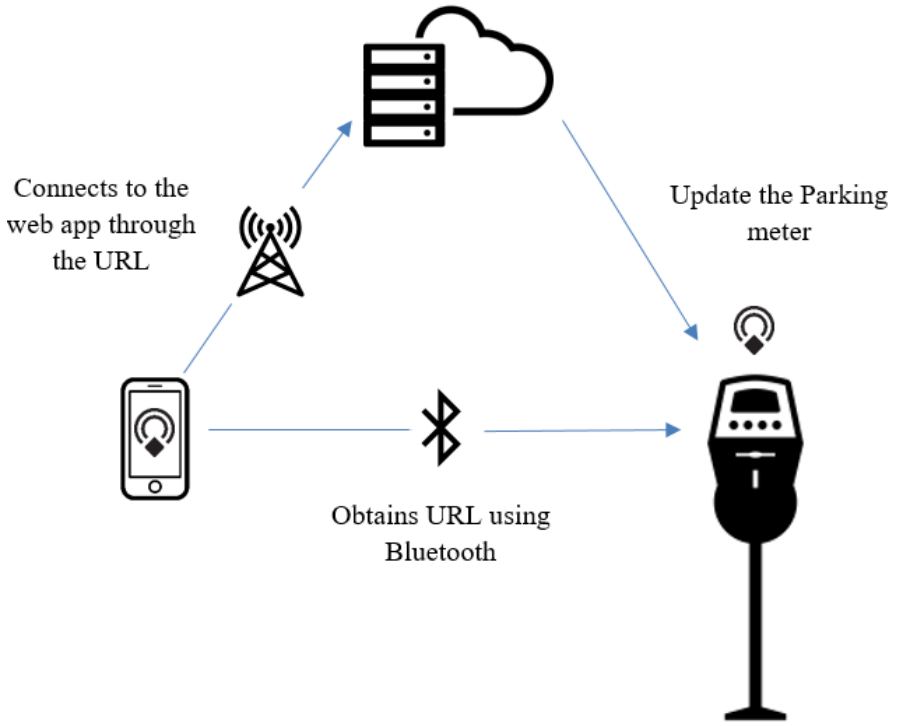


Figure 14 The Parking Meter Model

for a Parking Meter system. The system's description is as follows, the driver who had just parked his/her car must reserve the parking place for a period of time, therefore, by using a webpage stored on the Parallella cluster, the driver can obtain the URL from the beacon streaming it and integrated inside the parking meter in order to make the payment or extending the time of parking by paying extra money through the web page hosted on the cluster. The figure below is showing the implementation of the cluster and its role in the proposed model. Definitely, is still a theoretical proposal but since it is possible to

form a cloud server for a limited usage from the Parallella boards which can be comparable to a cloud server formed by Raspberry Pi boards.

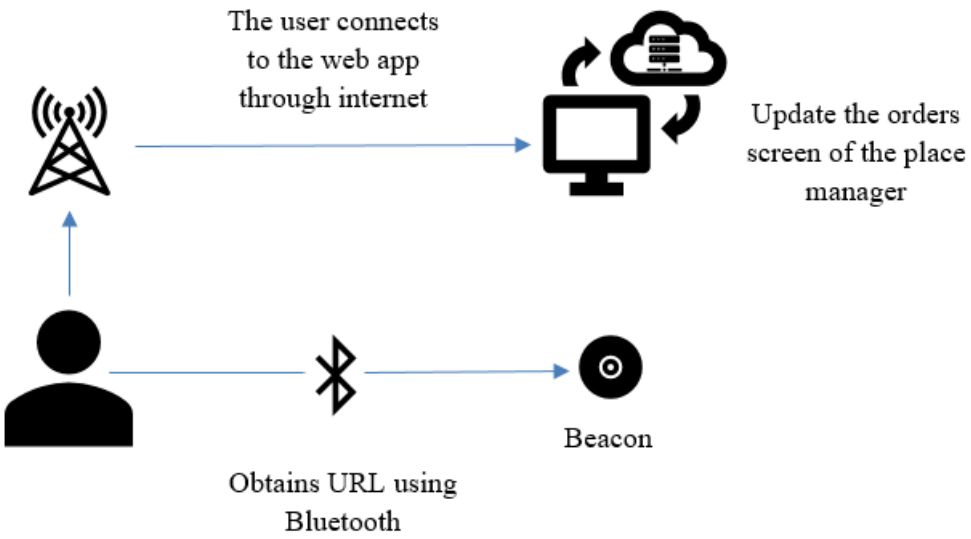


Figure 15 The restaurant model

6.3 Self-Order Model

This proposal can be considered as another scenario of physical web because it is based on the urls continuous streaming by beacons (iBeacon, Eadystone. etc.) but this time inside a restaurant where a customer can make an order for food through a small app or web page hoted on the Parallella Cluster. The URL for the ordering page can be obtained from the beacons install inside the place, and one may ask, how to distinguish between each customer's order and that can be answered by inserting the number of the table in the url in order to differentiate between each order and that would be filled by the customer itself (including the table's number). The benefit of this model is to make the people inside the restaurant feel free to order whenver they want and having a quick response to their orders where this service may replace tens of workers who take the customers meal orders.

6.4 A Secondary Web Server

Theretically, it is possible to have Parallella Cluster under its current circumstances and its configurations, as an additional web server which can handle queries in order to help the main servers in responding to all the clients' requests and that may prevent the servers from crashing or at least from the service cut. Of course the Parallella Cluster's capacity won't be the same as the main servers but still function as a webserver and respond to

many queries. Also it can provide in cases of fault tolerance or redundancy where one of the main servers is down or needs maintenance, in that case Parallella Cluster may provide the same service to the clients but it may not be as the same quality as the main web server.

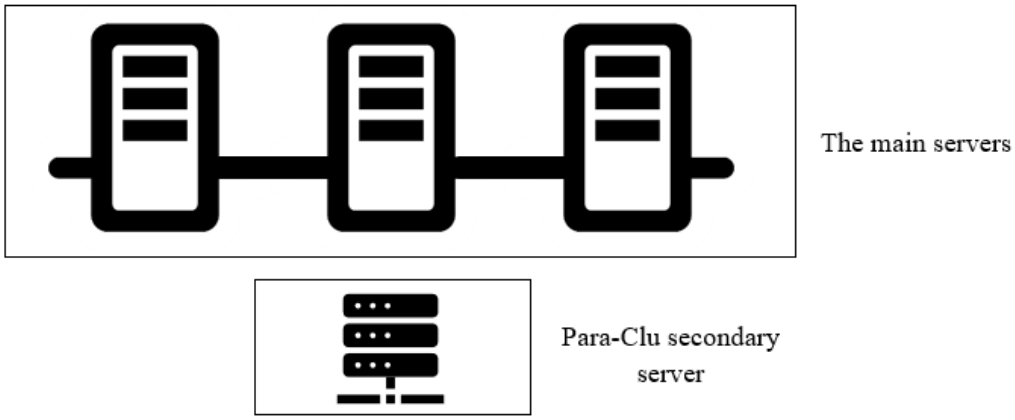


Figure 16 Parallella Cluster as a Secondary Server

CHAPTER 7

POWER CONSUMPTION MEASUREMENTS

Finally, in the power measurement section, the work of showing the energy efficiency of the cluster will be demonstrated by getting the power readings from the Parallella boards. In order to perform this operation, another extra materials are needed to get the power readings from the cluster such as the Hall Effect *ACS712* current sensor and an *Arduino UNO* board, the general setup can be described by just plugging the current sensor and connecting the positive and negative power supplies to its corresponding entries in the sensor. The ACS712 sensor has three outputs represented as VCC, VOUT & GND where the first is for getting the voltage reading, the second for the analog connection to the arduino and the last is the ground pin. Some of the features of ACS712 are:

- 185 mV/A current sensitivity (for a 5V supply).
- 5 V supply operation
- Output voltage for AC & DC currents.

Concerning the arduino board, there are a lot of candidates (variants of arduino) for this job, but in this setup, the arduino UNO is used. The choice of this board simplified many complications of adding some other intermediate parts in order to connect it to the ACS sensor.

Putting all the available parts together in the correct implementation makes everything ready for starting the power monitoring, however, before that, there must be some points and simple mathematical equations to mention here so the power consumption reading process can be performed in a smooth and correct way [6].

7.1 Power equation

The power equations are given in terms of the Voltage *V* and the current in amps *A*.

- $\boldsymbol{Pow = Volt \times Curr = V \times I}$ where *V* is the voltage and *I* is the current in Amps. The result of this equation is the power in *Watt*.
- $\boldsymbol{E = Pow \times T}$ where the power is in Watt, the *T* is the time in *hours* therefore, the energy's unit is *watt-hour* or *kWh*.

The following are the current (in amps) and power (in watt) readings. The Parallella cluster was put under *Siege Load Testing* in order to measure the performance of the Parallella web server under heavy *http* requests flow. Various numbers of concurrent users were tried such that the results will indicate at which level the cluster can still function and responds to the http requests sent by the *siege* utility.

The results showed that whenever the number of concurrent users increase, the possibility of having some failed transactions increase too, however, the amount of successfully answered requests (transactions) is considerable compared to the amount of power consumed by the cluster which is stated in the figure below.

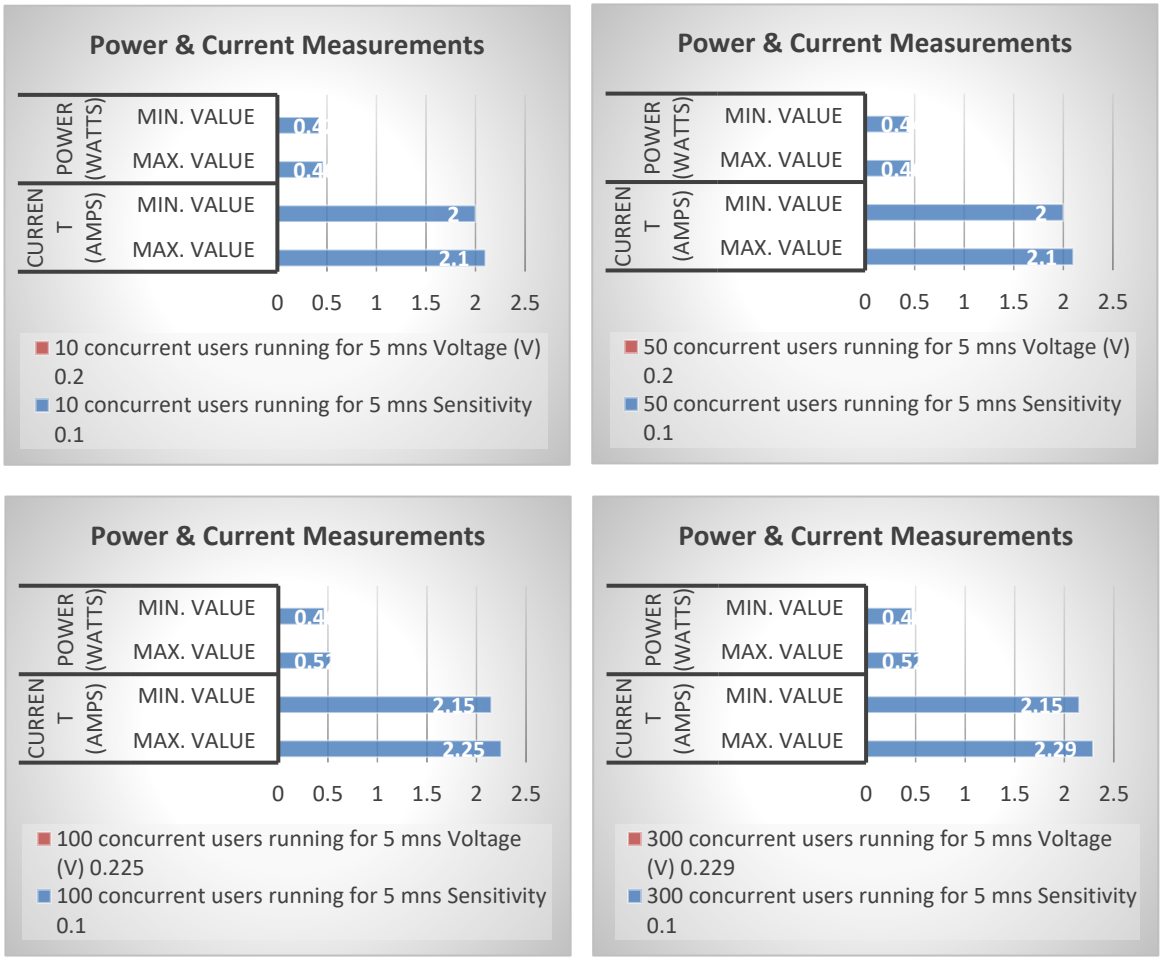


Figure 17 The Power & Current (*I*) readings for sensitivity of 0.1

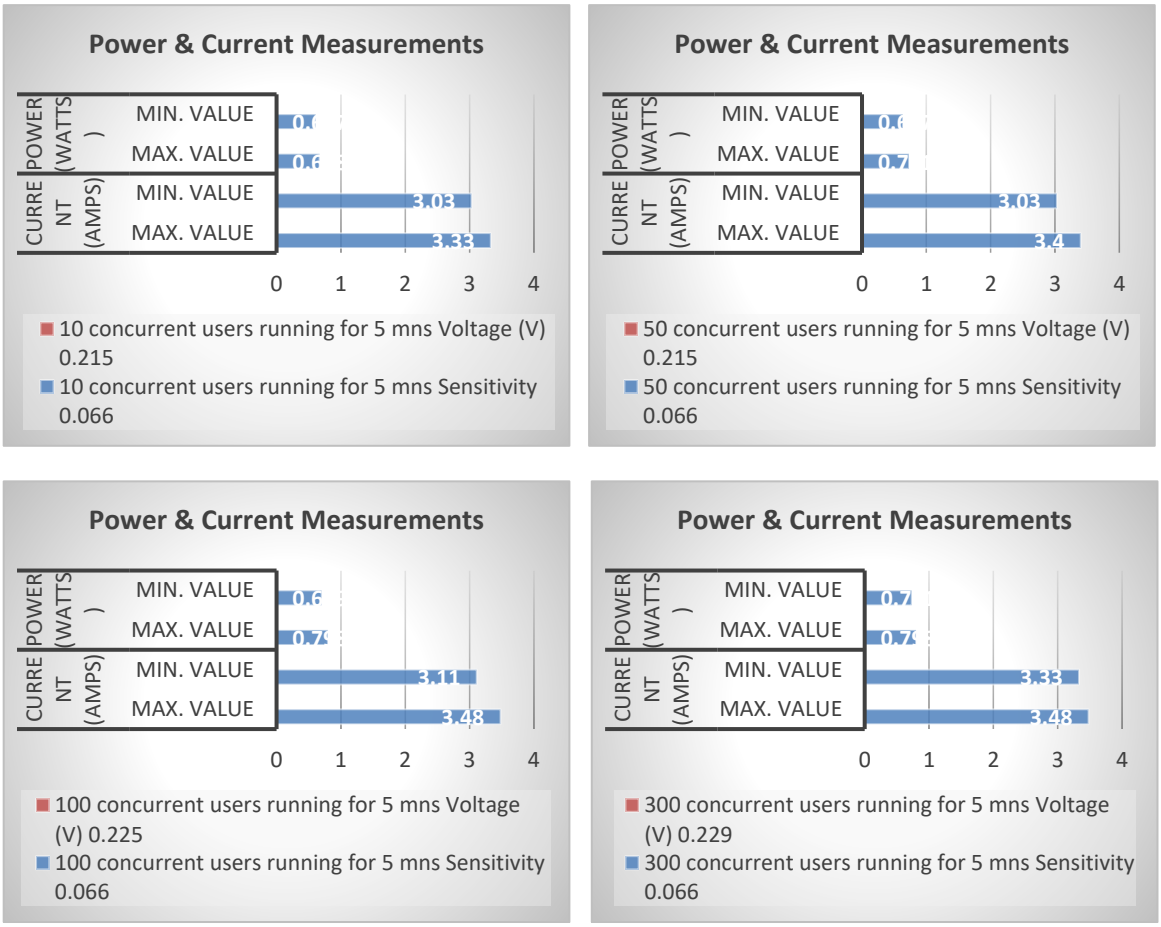


Figure 18 The Power & Current (I) readings for sensitivity of 0.066

Concerning the power ***P*** and current ***I*** readings when testing the Parallella cluster through the web application, both values were fixed, and no changes were spotted after changing the number of http requests, starting from 100 up to 10^5 requests, after such a number, the machine starts to slow down and sometimes freezes since it cannot handle this much active packet flows.

Time interval (5 min)				
Sensitivity = 0.1 mA/V				
Concurrent users N°	**Num. Transactions**	**Failed Trans.**	**Trans. Rate avg.** (trans/sec)	**Availability** (%)
10	11798	0	19.67	100
50	58971	0	98.36	100
100	116479	1	194.295	99.9999
300	128367	26	213.94	99.9997

Table 1 The number of successfully responded transactions for different number of concurrent users for sensitivity = 0.1 mA/V

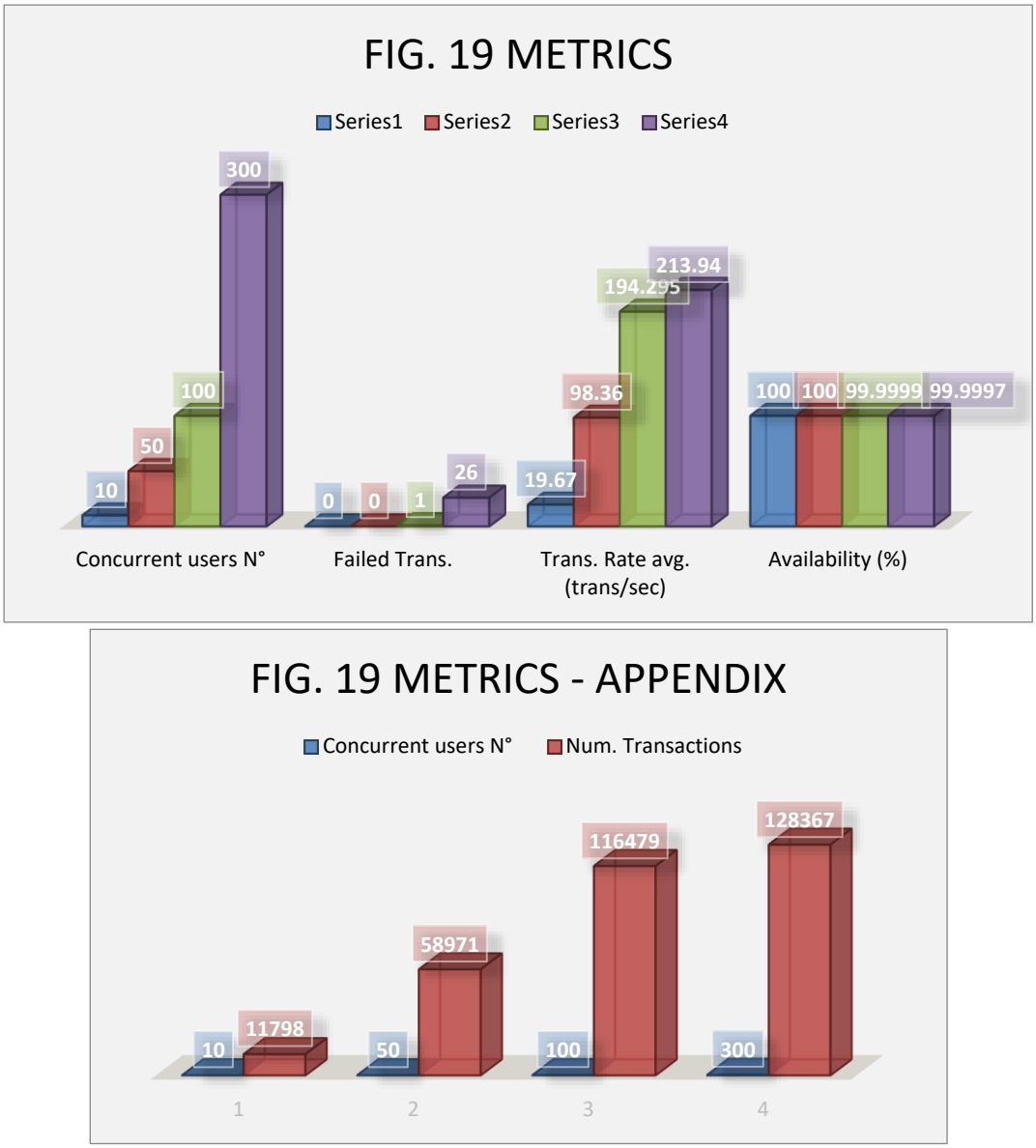
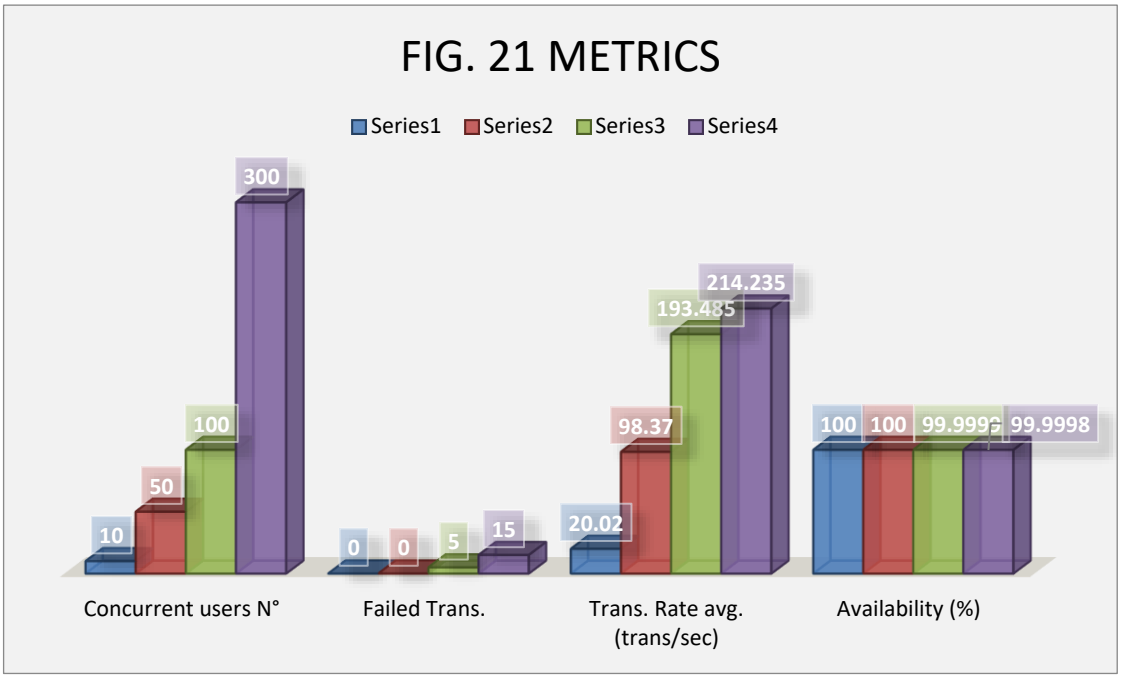


Figure 19 Data charts for the first table results readings

Time interval (5 min)				
Sensitivity = 0.066 mA/V				
Concurrent users N°	**Num. Transactions**	**Failed Trans.**	**Trans. Rate avg.** (trans/sec)	**Availability** (%)
10	11990	0	20.02	100
50	58988	0	98.37	100
100	115950	5	193.485	99.9999
300	128522	15	214.235	99.9998

Table 2 The number of successfully responded transactions for different number of concurrent users for sensitivity = 0.066 mA/V



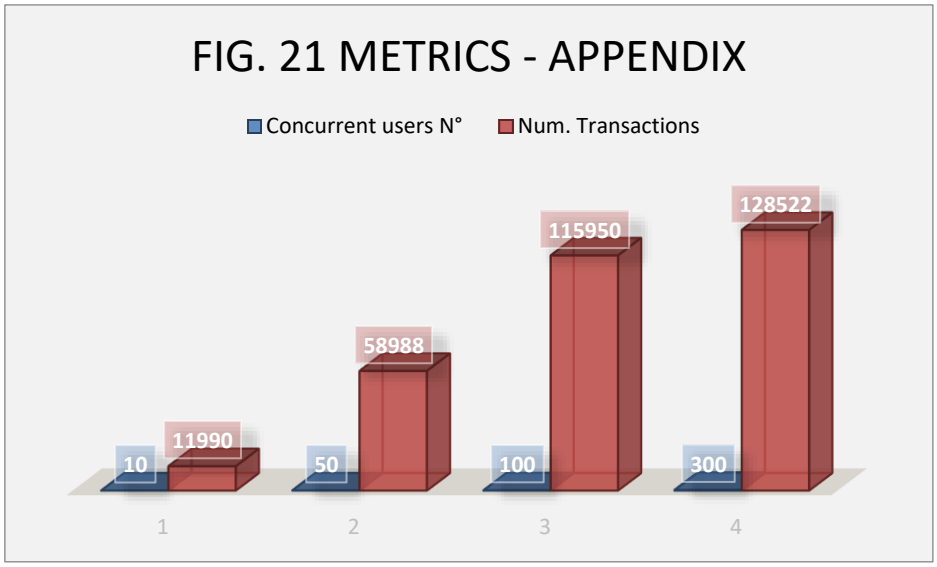


Figure 20 Data charts for the second table results readings

7.2 Comparing with similar boards

In order to give this project's a flavor of reality, a quick and brief comparison with other similar boards (in functionality and performance) is made in this subsection that shows the power consumption for each board's model in two different states (except for Parallella). The following power consumption readings were taken from ***Dramble*** which is a cluster of raspberry Pis boards powered by Kubernetes, Ansible, and Drupal 8 0. The following power consumption readings state the difference between the boards selected along with Parallella cluster in matter of power efficiency.

Board Model	Board's State	Power Consumption
Parallella P1601	Load Test with Siege (-c 300)	3480 mA (0.798 W)
Raspberry Pi 3 B+	Load Test with Apache Benchmark	950 mA (5.0 W)
Raspberry Pi 3 B+	stress --cpu 4 (400% CPU load)	980 mA (5.1 W)
Raspberry Pi 3 B	Load Test with Apache Benchmark	480 mA (2.1 W)
Raspberry Pi 3 B	stress --cpu 4 (400% CPU load)	730 mA (3.7 W)
Raspberry Pi 2 B	Load Test with Apache Benchmark	450 mA (2.3 W)
Raspberry Pi 2 B	stress --cpu 4 (400% CPU load)	400 mA (2.1 W)
Zero	HDMI off, LED off	80 mA (0.4 W)
Zero	HDMI off, LED off, USB WiFi	120 mA (0.7 W)
B+	Load Test with Apache Benchmark	180 mA (0.9 W)
B+	stress --cpu 4 (400% CPU load)	220 mA (1.1 W)
A+	Load Test with Apache Benchmark	80 mA (0.4 W)
A+	stress --cpu 4 (400% CPU load)	160 mA (0.8 W)

Table 3 Comparison of different boards' power consumptions under several states

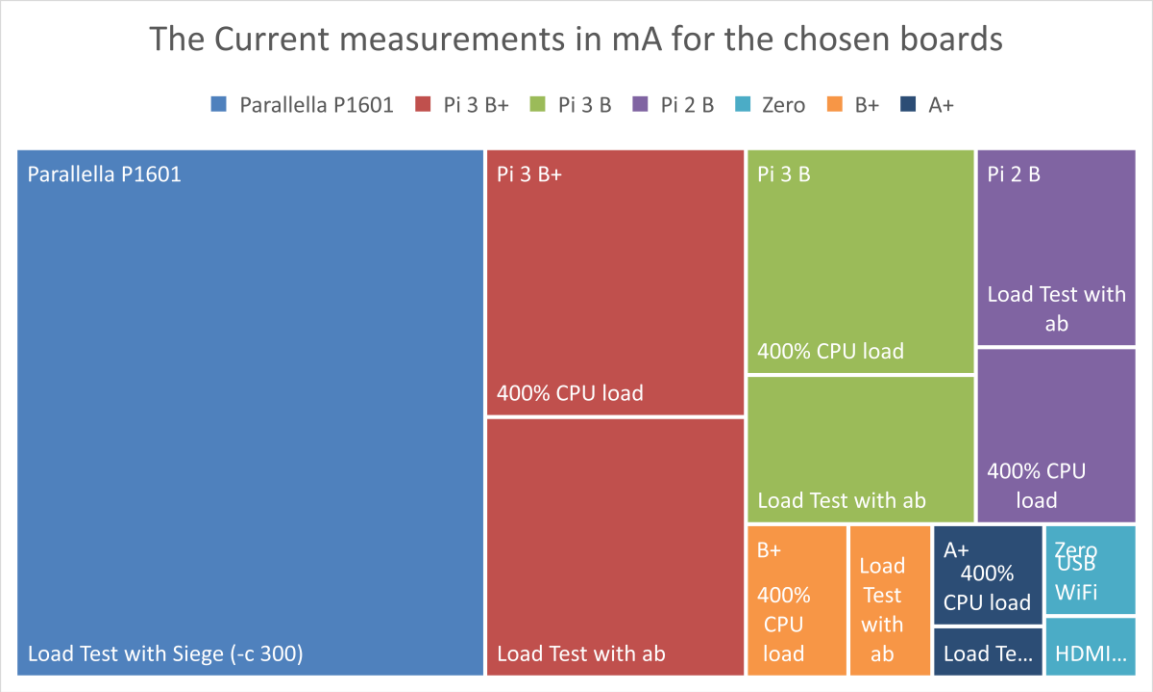


Figure 21 Classifying the boards according to their current consumption

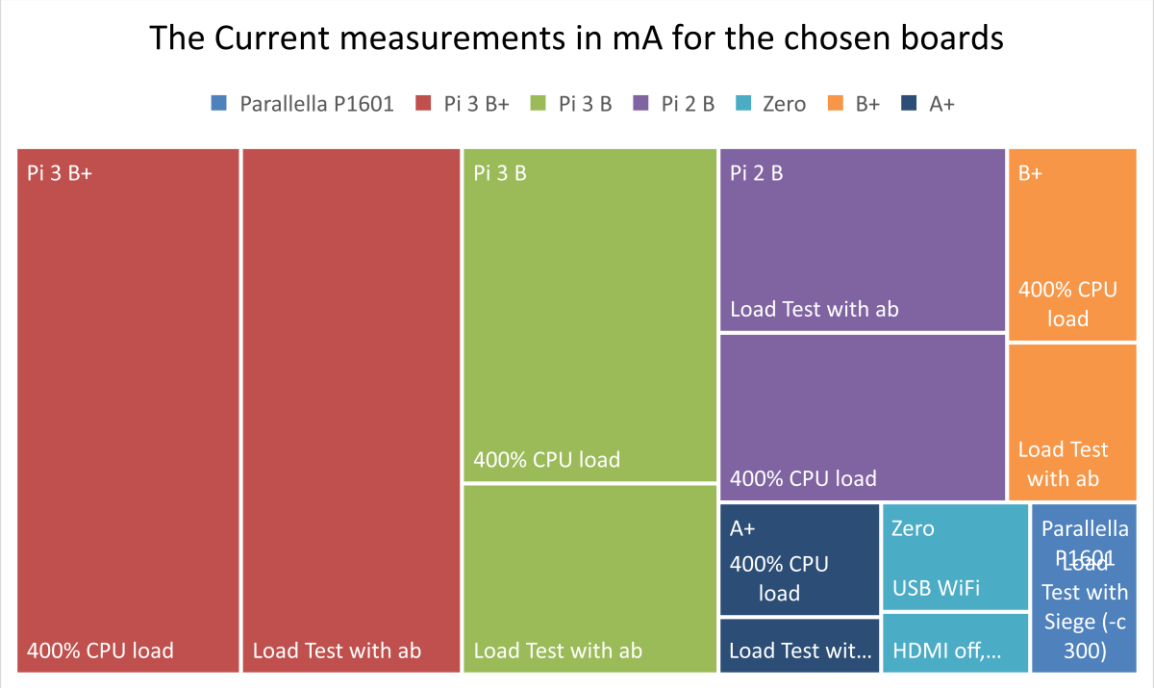


Figure 22 Classifying the boards according to their power consumption in Watt

7.2.1 Results discussion

The outcome that can be deduced from the previous tables and figures showing the power consumption for the Parallella cluster and other Raspberry Pis is that the Parallella cluster consume much less power than the other boards when tested under a stress load test, however, the important point that must be highlighted is that due to the incapability of using the Epiphany cores an the FPGA it became uncertain whether to trust these results since it seems an unfair comparison where the desired performance of the Parallella cluster was not achieved therefore, comparing it with other Raspberry Pis would be inequitable and somehow misleading, nevertheless, it is important to come up with a slight idea about theses boards' performances and their corresponding power consumption to decide whether our Parallella cluster is a power-efficient web server or not.

7.3 Summary

From the previously stated results showing different setups of testing the Parallella cluster under load testing tool and also through the web page made for it, an inference can be extracted about the power consumption of the Parallella-based web server which -as it was shown before- consumes power not greater than 0.8 watts when tested at its limits and that has proven the very first hypothesis about the Parallella Cluster web server being

power efficient and robust for handling *http requests* either from one source or different sources. Even though this project has suffered from several obstacles concerning the software side of it and the technical support for Parallella board has been withdrawn where no more feedback are given after 2017 but still this project can be considered as the starting point for any further projects based on Parallella either for solving its issues or building a new project starting from this one by benefitting from the information and the appendices presented in the thesis.

CHAPTER 8

CONCLUSION

In this project, building a cluster web server out of Parallella boards formed a challenge since there were scarce resources about it and the support provided by its manufacturers was limited. While working on the cluster, some of software requirements were already disappearing from the internet like the ubuntu distribution for Parallella which cost a considerable amount of time, however, the point that it was planned to be proven from the beginning was reached successfully and although all the blemishes in the boards (either Hardware or software) the Parallella Cluster was constructed and put to work as a web server for a standard web page stored in it which can be replaced by another web application or a simple web page, the reason for choosing a simple html web page was for benchmarking purposes.

There were some suggested scenarios for potential uses of the cluster, in addition to those, this web server cluster can be used for educational purposes for example, Parallel processes and algorithms and High-Performance Computing.

The built project can be considered as a baseline and a starting point for other projects willing to use Parallella boards since all the encountered problems are solved and their corresponding solutions are grouped in the appendices.

The Parallella web server cluster has shown after the experiments that the hypothesis of being power efficient and a low consumer of energy, it can be reliable to be used in small to medium tasks as proposed previously in the usage scenarios.

As a further work, unless someone adopt the Parallella project again and solves the board's fundamental problems like the Epiphany SDK issue, the Parallella board can turn into a waste of valuable resources, however, this project makes use of the available and possible hardware and software resources of the board and represents a good example of

using the Parallella boards as a web server. Again, the source codes for the whole project of Parallella web server are on GitHub[6].

[6] https://github.com/RoronoaZ/PARALLELLA.git

REFERENCES

[1] Adapteva. (n.d.). Retrieved from Adapteva..com: http://www.adapteva.com/epiphany-sdk/

[2] Apache. (n.d.). Apache Jmeter. Retrieved from https://jmeter.apache.org/: https://jmeter.apache.org/

[3] Barney, B. (2018, 06 25). Introduction to Parallel Computing. Retrieved from computing.llnl.gov: https://computing.llnl.gov/tutorials/parallel_comp/

[4] beacons-everything-you-need-to-know. (n.d.). Retrieved from www.pointrlabs.com: http://www.pointrlabs.com/posts/beacons-everything-you-need-to-know/

[5] Eddystone. (n.d.). Retrieved from https://github.com/google/eddystone: https://github.com/google/eddystone

[6] edie.net. (n.d.). edie.net. Retrieved from edie.net: https://www.edie.net/news/6/10-million--comatose--IT-servers-worlwide-wasting-electricity/

[7] Forbes. (n.d.). Retrieved from Forbes.com: https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#3426c0292d51

[8] HPC - High Performance Computing. (n.d.). Retrieved from https://hpcc.usc.edu: https://hpcc.usc.edu/support/documentation/message-passing-interface/

[9] KAHN, G. (1974). The Semantics of a Simple Language. 471-475.

[10] Kickstarter. (n.d.). Retrieved from Kickstarter.com: https://www.kickstarter.com/projects/adapteva/parallella-a-supercomputer-for-everyone

[11] Kruger, M. J. (2015, 11 23). Building a parallella cluster. Rhodes University.

[12] Load Testing and Benchmarking With Siege. (2012). Retrieved from ServerWatch: https://www.serverwatch.com/tutorials/article.php/3936526/Load-Testing-and-Benchmarking-With-Siege.htm

[13] openmpi. (n.d.). Using OpenMP – Portable Shared Memory Parallel Programming. Retrieved from OpenMP: https://www.openmp.org/

[14] Oracle. (n.d.). Oracle, Java Documentation. Retrieved from Oracle: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

[15] Parallella. (n.d.). Retrieved from parallella.org: https://www.parallella.org/board/

[16] Parallella. (n.d.). Retrieved from Parallella.org: https://www.parallella.org/docs/parallella_manual.pdf

[17] Parallella. (n.d.). Retrieved from Parallella.org: ftp://ftp.parallella.org/ubuntu/dists/trusty/image/

[18] Parallella-forums. (n.d.). Retrieved from Parallella: https://parallella.org/forums/viewtopic.php?f=10&t=1440

[19] Proulx, P. (n.d.). barectf 2: Continuous Bare-Metal Tracing on the Parallella Board. Philippe Proulx. Retrieved from Lttng.org: http://lttng.org/blog/2015/07/21/barectf-2/

[20] STACKPATH. (n.d.). Retrieved from http://www.maxcdn.com: https://www.maxcdn.com/one/visual-glossary/websocket/

[21] Sunrom Electronics | Technologies. (n.d.). Retrieved from Sunrom: https://www.sunrom.com/p/current-sensor-20a-acs712

[22] Woodard, L. (2013, 06 11). Introduction to Parallel Programming. United States of America

[23] dramble. (s.d.). pidramble. Récupéré sur Raspberry Pi Dramble: http://www.pidramble.com/wiki/benchmarks/power-consumption

APPENDIX A

SOLVING THE SPACE ALLOCATION PROBLEM

There was a problem when downloading new libraries to Parallella, the board was returning an error about insufficiency in the memory space and that was due to the space allocation after the installation of Ubuntu where 6.84 GB is reserved for the operating system and the rest was labeled as “Unallocated” which means that it is not possible to use it as free space even though it is a free space indeed but not prepared for utilization. The following steps supported with figures will demonstrate how to fix this issue which would save time of looking in others sources.

First, this figures shows how the space allocation looks after the setup of the operating system inside the Parallella board.

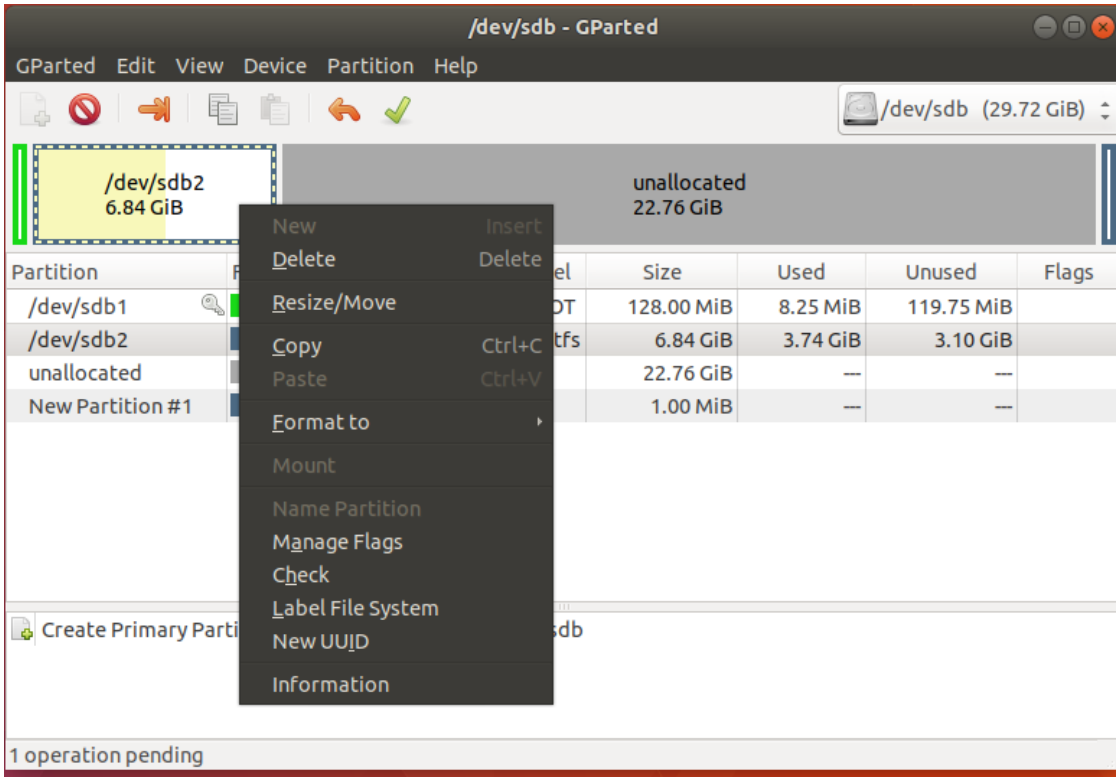


Figure 23 The unallocated space inside one Parallella board

The following step is to unmount the OS partition but the operating system isnt going to be deleted and a no re-installation is needed again. After that, it will be possible to re-allocated the free space as figure 15 shows below.

The goal is to expand the space of the operating system's partition by using the new allocated partition (#1). Therefore, to do that it is needed to only resize the new partition to 1 MB and after that there will be enough space to use for the OS partition.

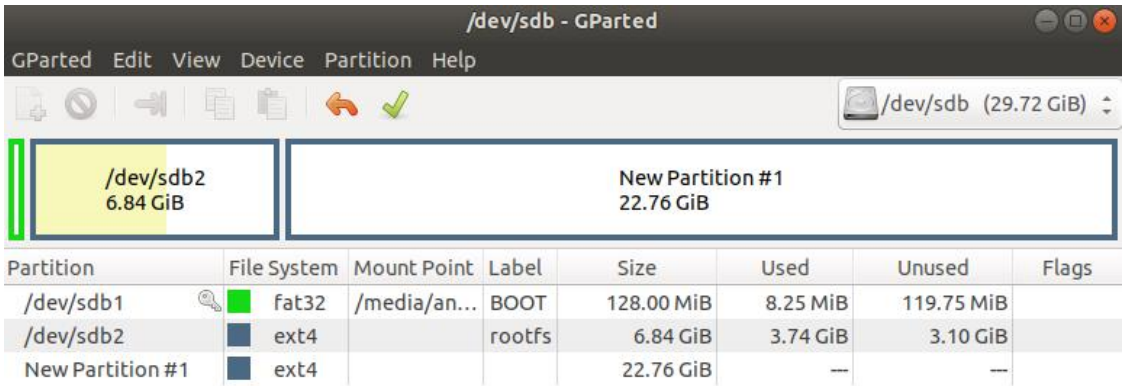


Figure 24 The new partition after unmounting the partition

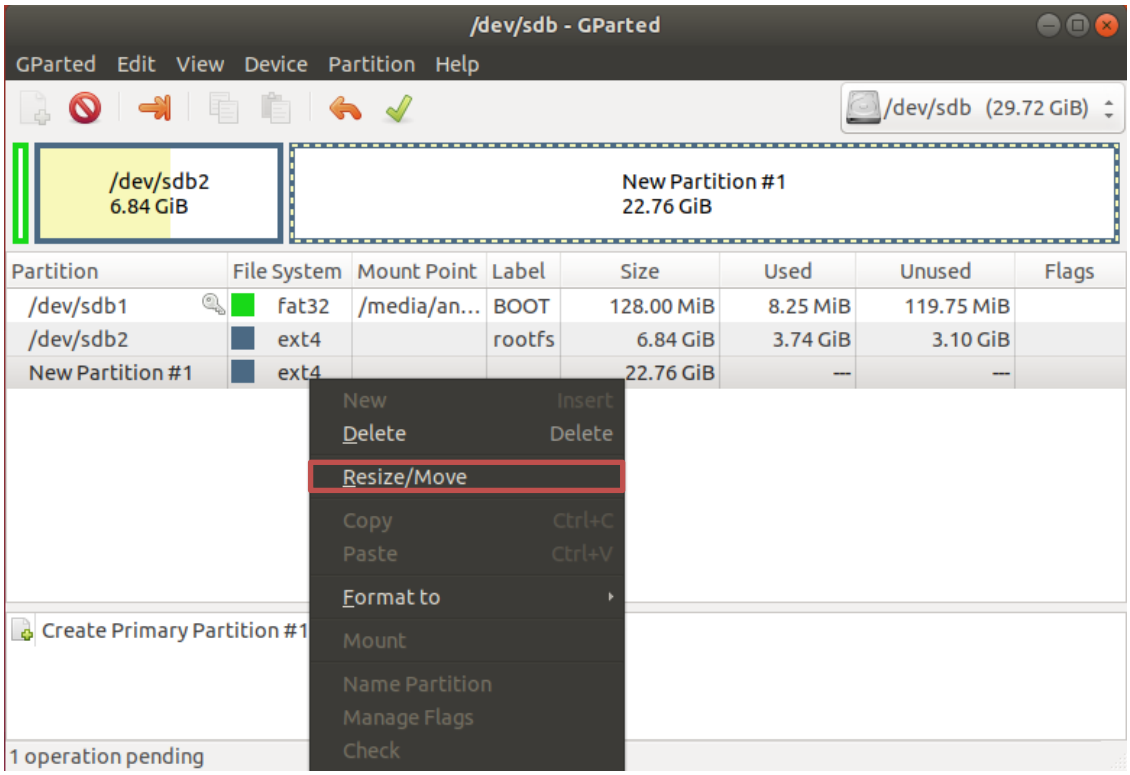


Figure 25 Resizing the new partition to 1MB

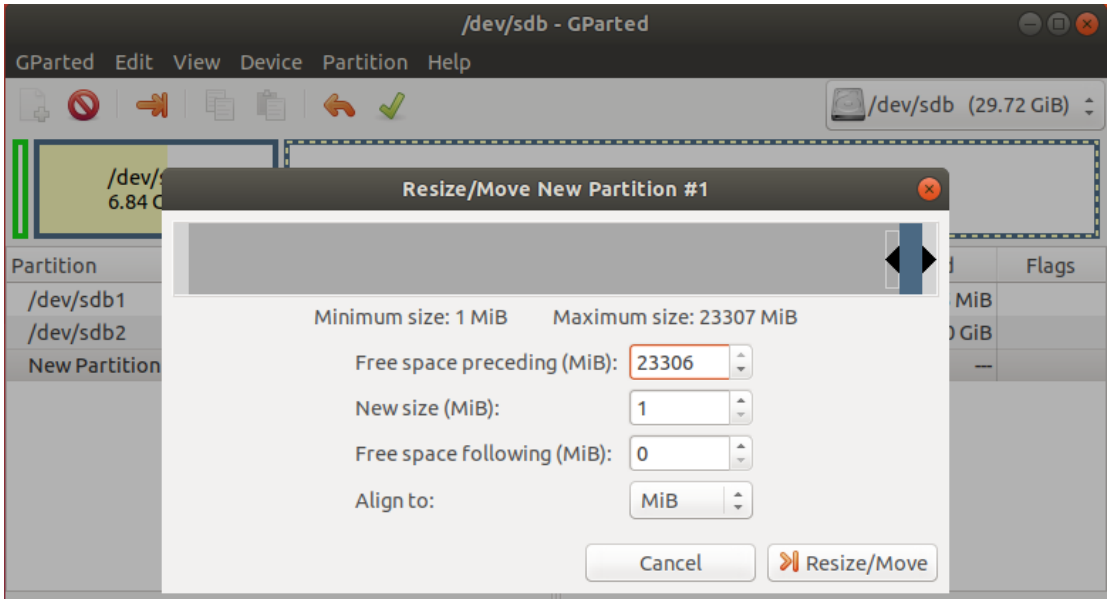


Figure 26 Resizing the new partition to 1MB - 2

The next figure is just an intermediate step to give more details about the resizing process Then, to expand the size of the operating system's partition, just drag the right edge of the partition to the right-most side in order to increase its size. The final step would be confirming all the changes that have been made so far which may take several seconds.

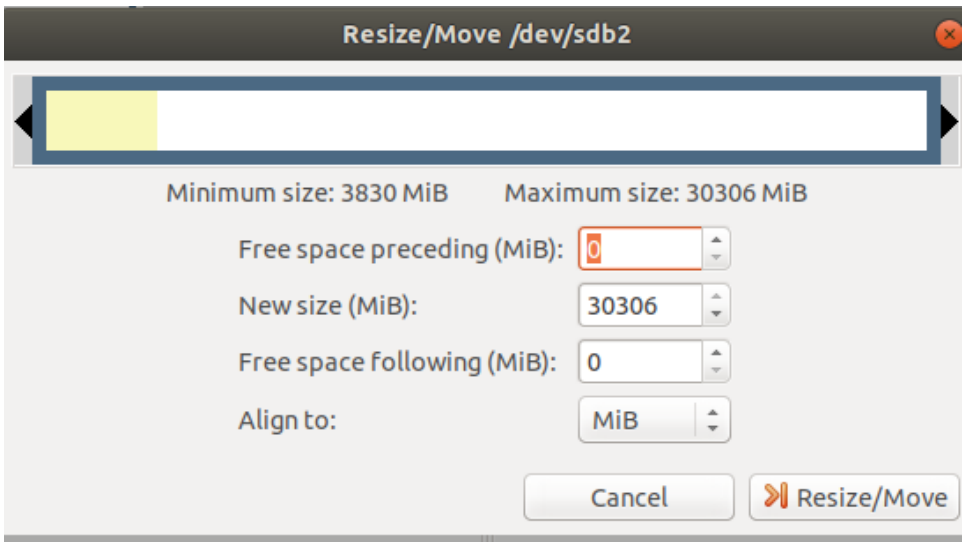


Figure 27 Expanding the size of the OS partition

Finally, the problem has been solved and the operating system installed in the Parallella board has sufficient memory space to install the required libraries in further usages.

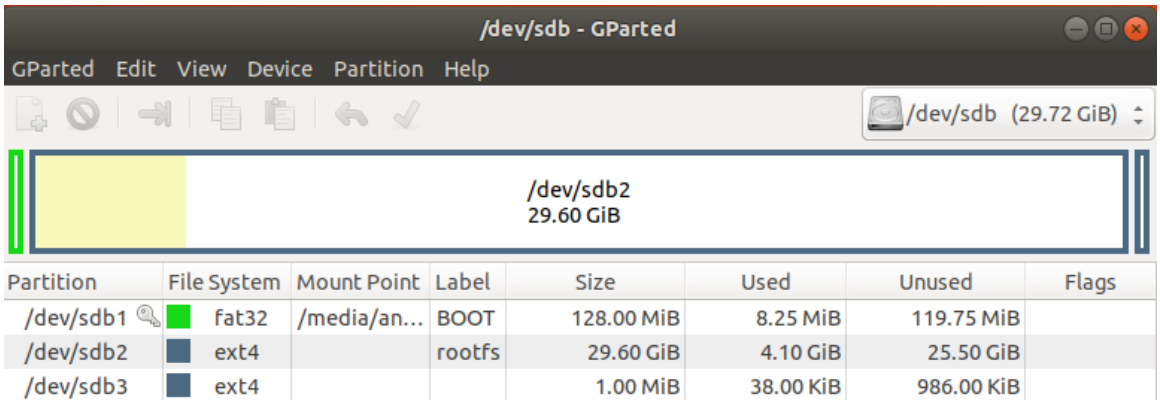


Figure 28 The final result of the allocating process

APPENDIX B

EPIPHANY SDK BUILDING UNSOLVED PROBLEM

Building the SDK which contains the main libraries responsible for the functioning of the Epiphany cores and the FPGA was not a successful process even after trying different Ubuntu OS distributions and installing only the required libraries, however, it is not possible since there are many dependencies between them. Nevertheless, it is substantial to show the details of this error such that further projects may follow easily the traces of this issue and perhaps solving it.

After downloading the SDK to Parallella and follow the steps of building it, the error will pop up after a while during the build of the toolchain, this process will take some time (sometimes aroun 3 hours) since the build process is formed of two sub-processes which are downloading the required libraries and then building them, the download phase has no problem, only when the system reaches the stage of building the whole process collapse.

Additionally, when trying to setup the libraries separately one by one, another error is shown on the screen as the figure below demonstrate when running the *./build-epiphany-libs.sh* command. It is always possible to trace the issue by reading the log file stored in */home/linaro/buildroot/logs/2016.11/* diretory.

Basically, the root of the problem which didn't allow exploring the full potential of Parallella boards was traced back to this point since many libraries plus thread-related ones, in addition to some compiling commands are depending on this SDK, therefore, for the record and for the upcoming projects, it is strongly recommended that any work on Parallella would start first from this point where there is an unsolvable problem concernsing the ESDK, especially for the projects aiming at making use of the epiphany cores and the FPGA, thence, as a recommendation from this project, the first task should be solving the ESDK library building issue taking into consideration that the support from

Parallella manufacturers is poor and their feedback about any reported problem barely exists.

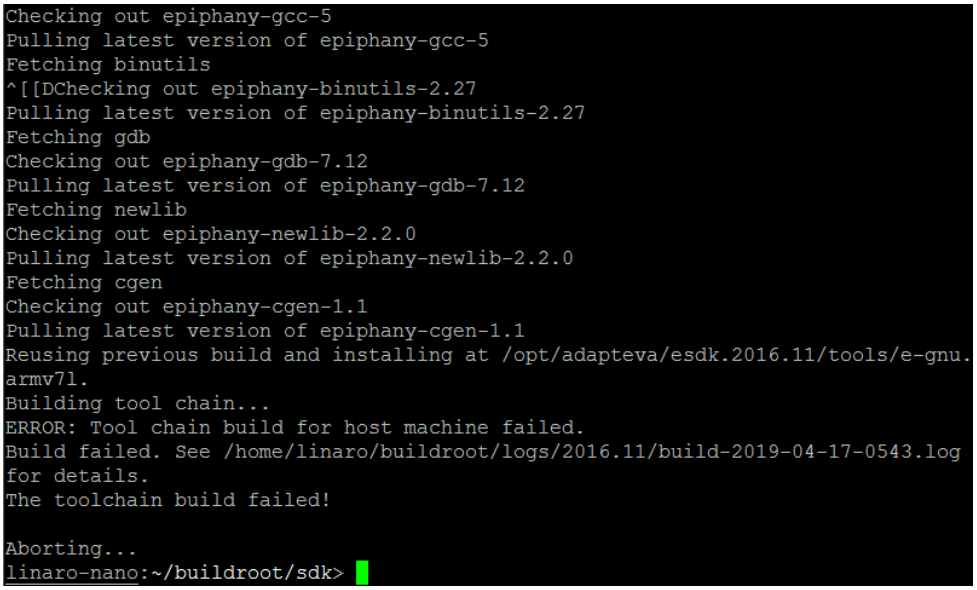


Figure 29 Building the epiphany libraries separately (failed)

The figure-27 shows the point in which the building process is aborted due to the failure in building the toolchain for host machine.

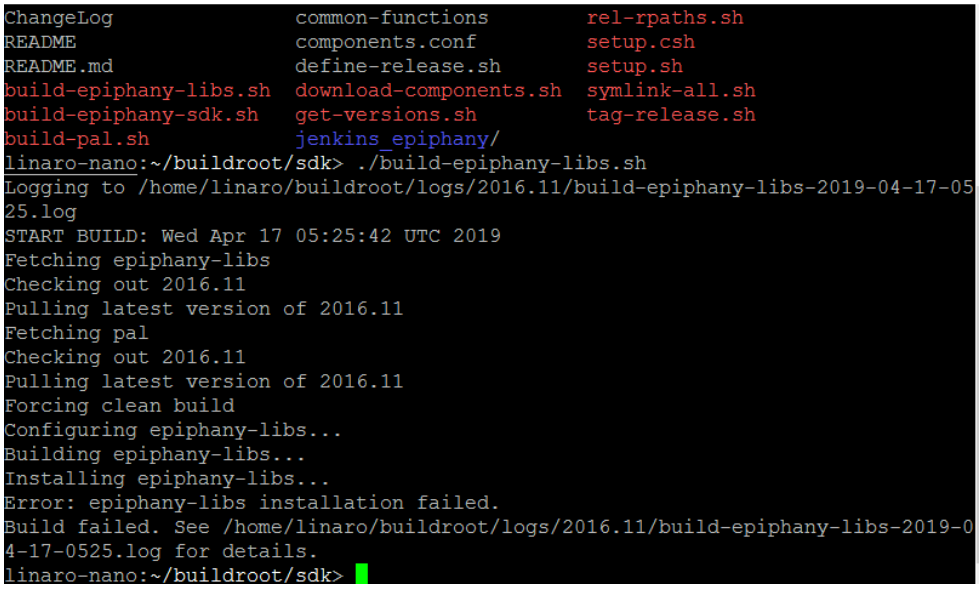


Figure 30 The ESDK building error

The figure-28 is showing that even the attemption to install and build the epiphany libraries independently from building the whole SDK is also not possible and returns an error for basically the same previous error shown in figure-27.

APPENDIX C

THE CLUSTER WORK DEMONSTRATION

This appendix actually exhibits the work of Parallella Cluster by showing the way of responding to the http requests sent from the simple web application. Actually, the queries are sent to both boards' (the working ones) IP addresses, however, in order to send these requests from outside the network, there is another way to do it by using a VPN connecting the person from outside the network where the cluster is having its IP address from and then access the web page in order to test the cluster. In this case, where the cluster is connected to the university's network, it would be a simple and secure process to connect to the same network of the cluster through the university's *virtual private network*.

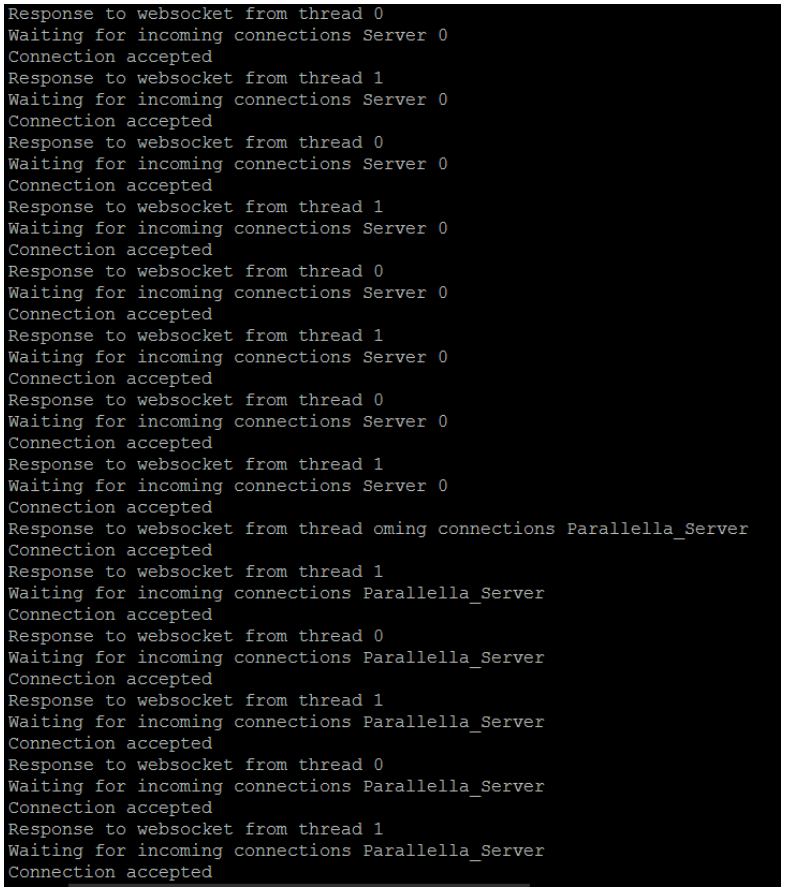


Figure 31 Responding in parallel to the requests sent from the web page

In a nutshell, the following figures show the way of the cluster's working and it is responding to the *http requests* sent from the web page which can be accessed by any one as long as it is hosted on the same network, or in another network but by using the previous VPN method.

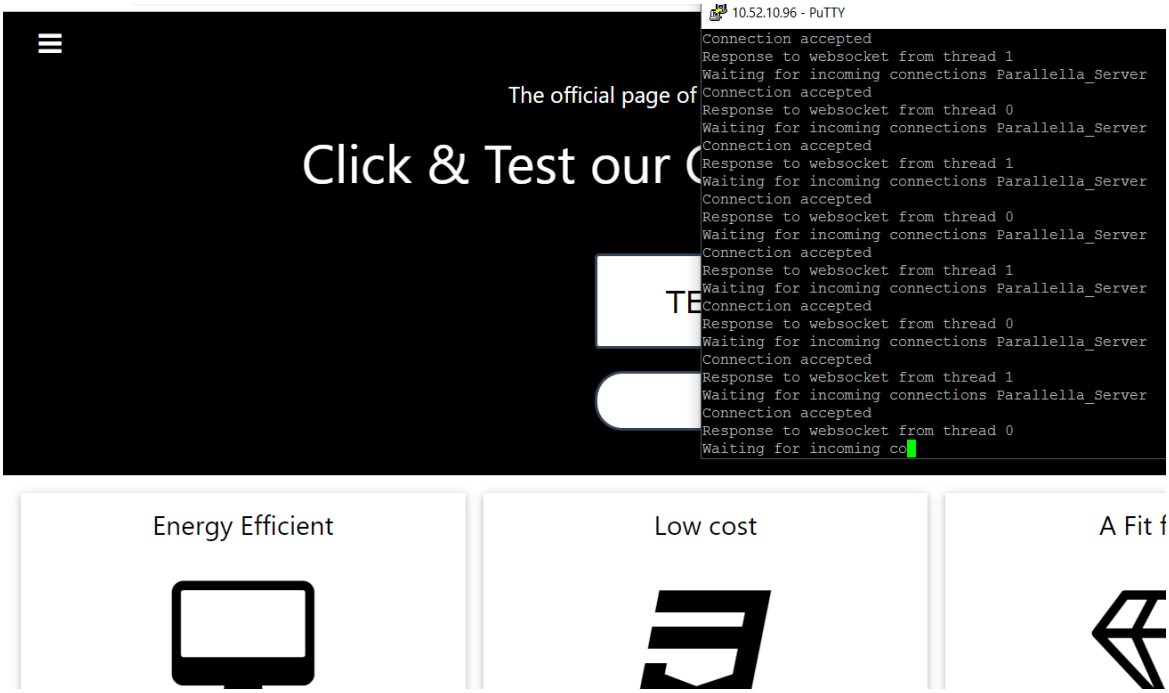


Figure 32 The web page and the process of receiving the http requests by the cluster

The figures show the messages of successfully responding to the *http requests* sent by the websockets from the web page, a full demonstration by video is also available which also shows the whole process of testing the cluster.

As it can be noticed, there are only two boards respongding to the queries (Server 0 & Parallella_Server) instead of 3 as stated previously and the reason behind that is due to some problems encountered while either putting all the boards together or assiging the IP addresses, one of the boards has a missing part responsible for cooling down the processor and the other board obviously has problem concerning its own ethernet interface which doesn't acquire any IP address and therefore it becomes impossible for this board to be a part of the cluster since it won't be able to receive and/or respond to any queries.