„POLITEHNICA” UNIVERSITY OF BUCHAREST

ETTI-B DOCTORAL SCHOOL

Decision No. …...…..... **from** ….......……

Hardware Offloading for Energy-Efficient Distributed Computing

- Summary -

Accelerare Hardware pentru Sisteme de Calcul Distribuit Eficiente Energetic

- Rezumat -

by **Eng. Ovidiu Plugariu**

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in Electronics and Telecommunications

DISSERTATION COMMITTEE

President	**Prof. Dr. Ing. Gheorghe Brezeanu**	from	**Univ. Politehnica Bucureşti**
Advisor	**Prof. Dr. Ing. Gheorghe Stefan**	from	**Univ. Politehnica Bucureşti**
Reviewer	**Prof. Dr. Ing. Monica Dascalu**	from	**Univ. Politehnica Bucureşti**
Reviewer	**Conf. Dr. Ing. Cristian Molder**	from	**Academia Tehnică Militară Bucureşti**
Reviewer	**Conf. Dr. Ing. Iulian Rancu**	from	**Academia Tehnică Militară Bucureşti**

BUCHAREST 2019

Contents

1 Introduction

1.1 Thesis research field

"ARM servers are a research subject for the industry [1] and Academia [2], [3], [4] because they are energy-efficient, low-power devices designed for high-throughput workloads which are not compute-intensive. Modern general-purpose server processors of the x86 and x64 architectures are fast, but they sacrifice power and energy-efficiency" [5].

All data centers (Figure 1) use additional equipment for operating the computing machines: backup power supplies and communications connections, air cooling and fire suppression devices, security equipment [6].

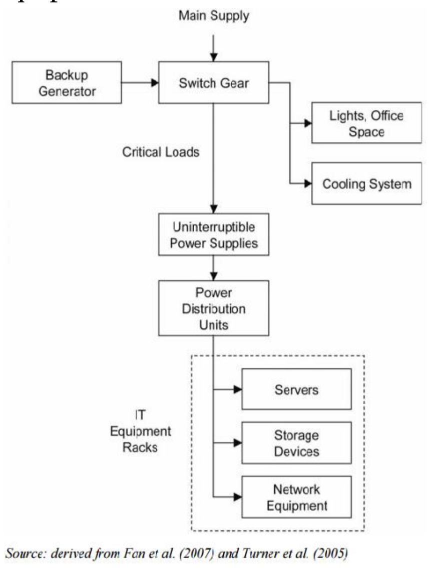
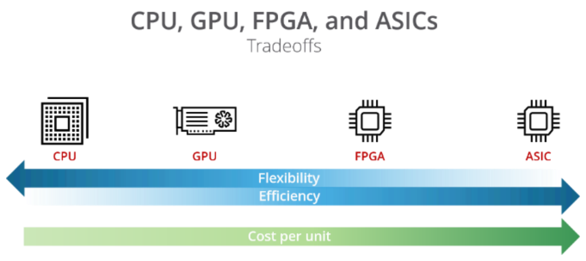


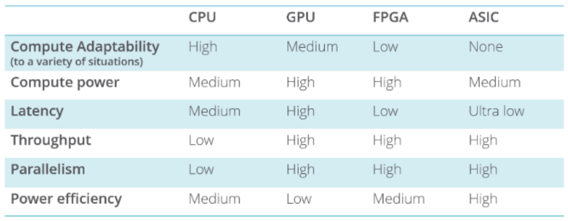
Figure 1 General Data Center architecture

Since the global amount of data is only going up, the best solution for obtaining more efficient data processing systems is to optimize the speed and energy consumption of the chips that process the payloads.

The most popular chip types used for processing data are CPU, GPU, ASIC and FPGA.



(a)



	CPU	GPU	FPGA	ASIC
Compute Adaptability (to a variety of situations)	High	Medium	Low	None
Compute power	Medium	High	High	Medium
Latency	Medium	High	Low	Ultra low
Throughput	Low	High	High	High
Parallelism	Low	High	High	High
Power efficiency	Medium	Low	Medium	High

(b)

Figure 2 Digital data processing chips performance and efficiency [7]

As we can see in in Figure 2 (a) and (b), FPGAS are a versatile choice for accelerating data processing while their only disadvantage is the long life cycle design.

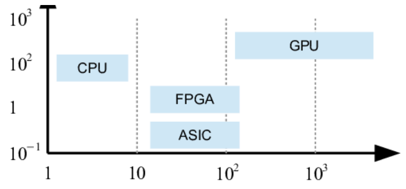


Figure 3 Power classification between CPU, FPGA, GPU and ASIC [8]

They are already used in wide-scale datacenters and a good example would be the Catapult project [9] and they have the best performance/Watt compared with GPUs and CPUs [8], [10] and [11].

1.2 Purpose of the thesis

“This thesis aims to study and implement an energy-efficient distributed computing heterogeneous cluster using the Hadoop framework with hardware-accelerated GZIP compression” [5].

The basic building clock of the cluster are the ZedBoard development boards which contain dual core ARM processors and programmable logic.

“The research will contain three main stages:

1. In the first stage, we will compare the processing speed and the energy-efficiency of an ARM based cluster with an x86 cluster using standard benchmarks in the Hadoop framework.
2. In the second stage of our research, we will develop the first open source, technology independent, low area GZIP FPGA core using the Verilog Hardware Description Language.
3. In the third stage, we will integrate the compression core in the Hadoop ZedBoard cluster and study the speed improvement and energy-efficiency of the cluster with FPGA accelerated compression” [5].

1.3 Content of the thesis

Chapter 2 will show all the steps for building an ARM-based heterogeneous cluster using ZedBoards.

Chapter 3 will compare the baseline performance of an Hadoop x86 cluster versus the ZedBoard cluster.

In Chapter 4 we will show some basic notions about data compression.

In Chapter 5 we will show the architecture of a hardware GZIP compliant compressor.

In Chapter 6 we will present how to integrate the GZIP core in the Hadoop cluster and will measure the speed and energy-consumption improvements.

In Chapter 7 we will present the final conclusions of this thesis.

2. Hadoop Cluster System design

Apache Hadoop is defined as "an open-source software platform for reliable, scalable, distributed computing. It allows distributed processing of large data sets across computer clusters using simple programming models" [12]. The Hadoop is composed of HDFS and MapReduce.

Typically, a Hadoop cluster (Figure 4) is compresed from a NameNode which is coordinating several DataNodes.

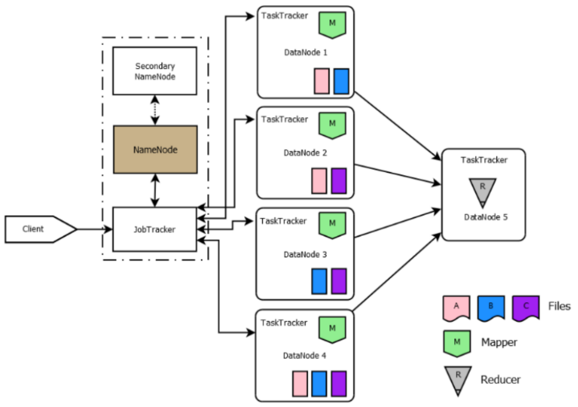


Figure 4 Hadoop cluster architecture [13]

2.1 Steps for setting up the distributed computing ZedBoard cluster

2.1.1 System Design

The block diagram of the implemented Hadoop cluster is present in Figure 5 .

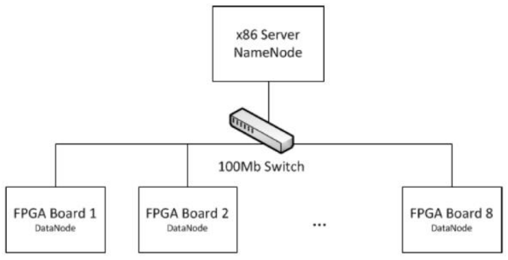


Figure 5 Architecture of the ZedBoard cluster

2.1.2 Cluster configuration

In this subchapter from [5] we describe all the necessary steps required to configure a distributed computing Hadoop ZedBoard cluster.

3 Evaluation of a distributing computing Zedboard cluster

In this chapter we will compare the baseline performance of two Hadoop clusters using the Terasort, TestDFSIO, MRBench and Wordcount benchmarks. One cluster is built using x86 machines (Figure 6) and the other one is built using ARM-based nodes.

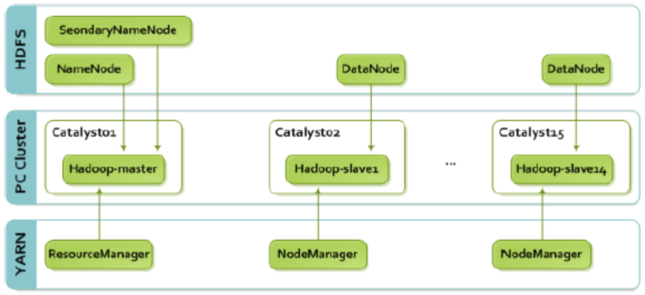


Figure 6 Hadoop components in the x86 cluster [14]

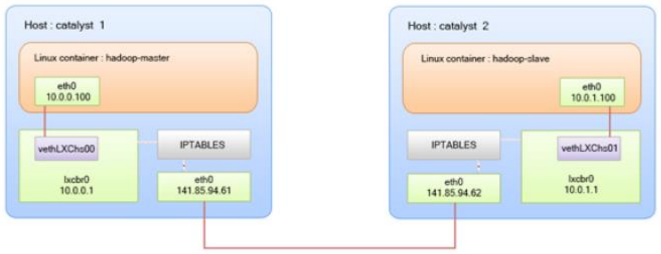


Figure 7 Architecture of the x86 cluster [14]

3.1 Wordcount

"The Wordcount benchmarking application counts all the occurrences of a word from a given set of text files in HDFS, using MapReduce. The input for this benchmarking application is the Wikipedia Corpus [15], totaling 9.6 GB of text. Wordcount is an evaluation of compound Hadoop performance (HDFS and MapReduce), but because the computation is exceedingly simple, i.e. repeated string comparison, Wordcount is more IO intensive than computationally intensive" [14].

"The ARM cluster processed the corpus in 19386 seconds (5.23 hours) while the x86 cluster completed Wordcount in 1982 seconds (0.55 hours). The x86 cluster performed almost 10 times faster compared to the ARM cluster. Because the data transfer speed of the SD card is an order of magnitude slower than a mechanical hard drive, HDFS performance is a bottleneck for the ARM cluster" [14].

3.2 Terasort

“Terasort is the most well-known Hadoop benchmark. The goal of Terasort is to sort 1TB of data (or any other size) as fast as possible in a distributed fashion and it it exercised both the HDFS and the MapReduce layers of the cluster. Since the computed algorithm is more complex than that of Wordcount, Terasort is focused more on computing performance and less on HDFS performance.” [14].

Figure 8 “illustrates the performance of the two clusters while executing Terasort. For both clusters, the time required to completely sort the input data is proportional to the size of the data-set. The x86 cluster is on average two times faster, for all dataset sizes.” [14].

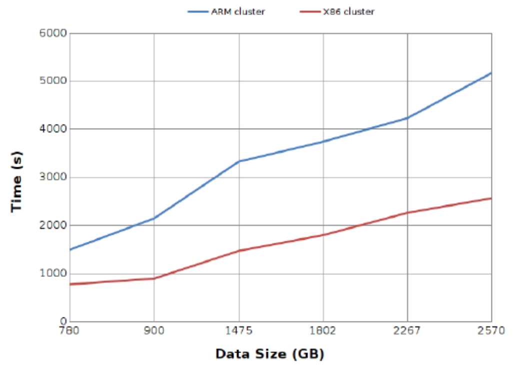


Figure 8 Terasort results [14]

3.3 MRBench

“MRBench executes a small Hadoop job in a loop for a large number of times, evaluating the performance of the job management mechanisms of the cluster. The number of jobs launched is the key parameter of MRBench and it focuses on the MapReduce layer as its impact on the HDFS layer is very limited.” [14].

“We executed MRBench with a number of jobs ranging from 1 to 100. Figure 9 presents the performance comparison of the two clusters on MRBench. Despite the fact that the JobTracker of the ARM cluster is a traditional x86 server, the under-powered ARM processors of the slave nodes take longer to respond to requests and overall MRBench performs 3 times better on x86 than on the ARM cluster” [14].

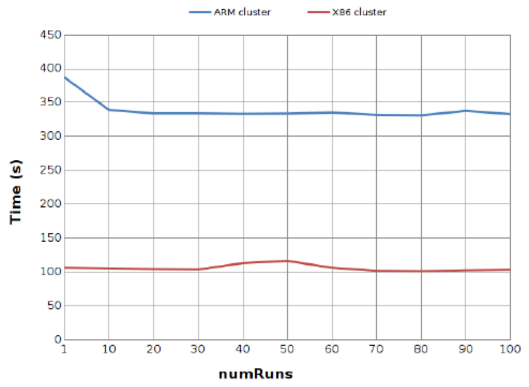


Figure 9 MRBench results

3.4 TestDFSIO

“Test DFSIO is a benchmark for read/write operations from and to HDFS. We performed several evaluations for a data volume of 2GB split equally into an increasingly larger number of files. The purpose of this evaluation is to identify how the data granularity is affecting the IO speed of the Hadoop cluster” [14].

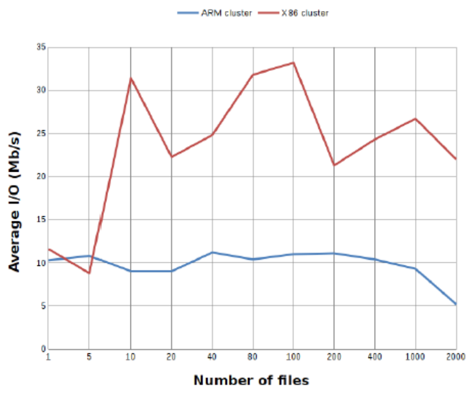


Figure 10 Test DFSIO Read Average I/O [14]

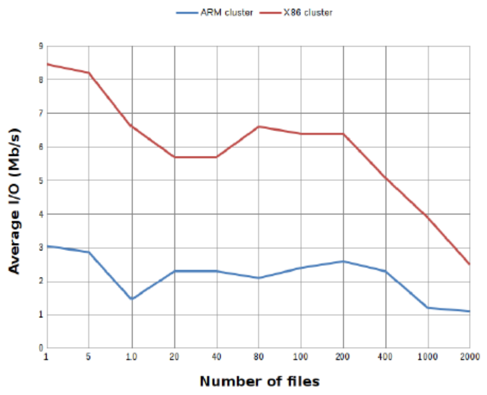


Figure 11 TestDFSIO Write Average I/O [14]

“Figure 10 illustrates the average read IO throughput of the two clusters under evaluation. Except when handling large files, the read throughput is 2-3 times greater on the x86 cluster, which utilizes mechanical hard-disks. Write performance is illustrated in Figure 11. For both clusters, write performance degrades when files become smaller, as the HDFS file management overhead becomes more time-consuming than the actual writes to the storage devices of the cluster nodes” [14].

Operation	Average I/O (MB/s)
x86 Cluster HDFS Read	23.47
ARM Cluster HDFS Read	9.7
x86 Cluster HDFS Write	5.95
ARM Cluster HDFS Write	2.1

Table 1 Average TestDFSIO Performance [14]

3.5 Power dissipation

“We evaluate the power dissipation of the ARM cluster in for providing a rough comparison of the two clusters for this important datacenter metric. We utilize an Ubiquity mPower Pro smart power outlet, with power measurement capability. The ARM and x86 cluster average power dissipation is recorded during the execution of the distributed computing benchmarks and presented in Table 2. We note that the power dissipated in the ARM cluster mostly by the name-node server, at 190 Watts. The worker ARM nodes dissipate a maximum of 20 Watts together. By taking into account the execution times for the Wordcount, Terasort, and MRBench benchmarks, we have also calculated the energy consumed by each cluster to compute the benchmarks. The ARM cluster is more energy-efficient on all benchmarks except Wordcount, which is IO dependant” [14].

Computing System	ARM Cluster	x86 Cluster
Power Dissipation (W)	210	1200
Wordcount energy (MJ)	4.07	2.38
Terasort Energy (MJ)	0.73	1.92
MRBench Energy (MJ)	0.07	0.12

Table 2 Power Dissipation [14]

3.6 Conclusions of initial cluster measurements

Despite the ARM cluster has slower IO it “dissipates five times less power than the x86 cluster and is more energy-efficient on most benchmarks.” [14].

4 Compression techniques and compression algorithms

“One of the most used features of Hadoop is data compression because the Big Data volumes are very large and would require many resources to be stored. Hadoop supports many popular compression algorithms like BZIP, GZIP, Snappy and LZO” [5].

“Data compression is the process of converting an input data stream (the source stream or the original raw data) into another data stream (the output, the bitstream, or the compressed stream) that has a smaller size. A stream is either a file or a buffer in memory.” [16].

4.1 Theory for data compression

“Data compression is performed by changing the data representation from long codes to short ones, using the following rule: short codes are assigned to symbols or phrases that occur frequently and long codes are assigned to the symbols that are less repetitive. The redundancies depend on the type of each processed payload (e.g. text, video, ...) which is why some compression methods are best only for specific types of data.

These are the 3 major compression categories:

1. *run length encoding (RLE)*
2. *statistical methods*
3. *dictionary based methods (Lempel-Ziv)"* [5]

"The compression ratio *CR* of a data source is defined by Equation 1." [16]

$$CR = \frac{size\ of\ output\ \ stream}{size\ of\ input\ stream}$$

Equation 1 [16]

"The compression gain *CG* of a data source is defined by Equation 2." [16]

$$CG = 100 log_e \frac{reference\ size}{compressed\ size}$$

Equation 2 [16]

4.1.1 Huffman Coding/Decoding

"One of the most used methods for compressing data is Huffman coding which creates better codes than the algorithm created by Shannon-Fano [16]. The Huffman method creates the codes starting from the right to the left bit after it builds the list of alphabet symbols sorted in the descending order of their probability of occurrence. The algorithm starts from bottom to top by assigning bit values to the symbols with the smaller probabilities and advances to the more frequent symbols by creating intermediary values. The algorithm is considered complete when all symbols are reduced to one intermediary symbol. After this, the tree is traversed from the end to the beginning to generate the Huffman code for each symbol" [5].

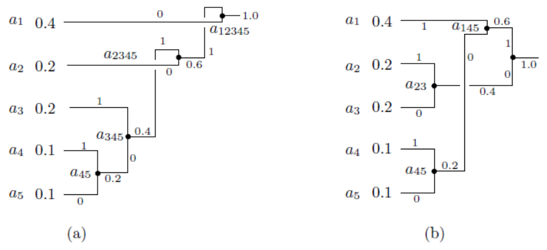


Figure 12 Huffman code example [16]

4.1.2 Dictionary methods

"The dictionary based methods are used to encode each symbol from a string with a ***token*** from a dictionary. The dictionary can be either static or dynamic. The static dictionary has some preset values while the dynamic version is updated with symbols as the input stream is processed" [5].

"Dictionary methods are very popular between the practical compression algorithms like LZ77, LZMA, LZW etc." [17] .

4.1.3 LZ77 (Sliding Window)

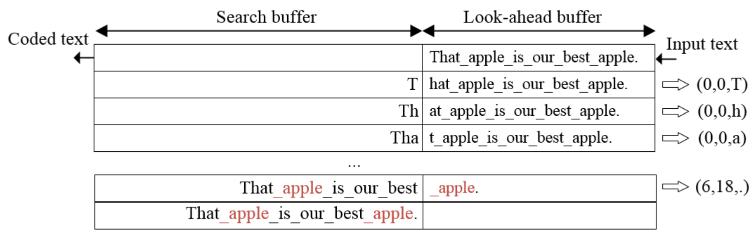

Search buffer
Look-ahead buffer
Coded text
Input text
That_apple_is_our_best_apple.
T hat_apple_is_our_best_apple.
(0,0,T)
Th at_apple_is_our_best_apple.
(0,0,h)
Tha t_apple_is_our_best_apple.
(0,0,a)
...
That_apple_is_our_best _apple.
(6,18,.)
That_apple_is_our_best_apple.

Figure 13 LZ77 sliding window example 1

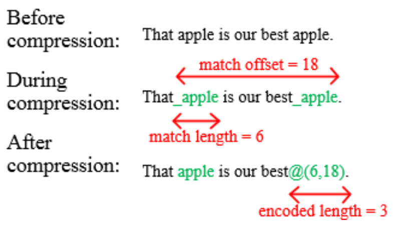


Figure 14 LZ77 sliding window example 2

"The core algorithm of GZIP, usually called LZ77 [18], is based on using a piece of the previously processed input data as the searching dictionary and to replace repeating parts of text with pointers to the previous text" [5].

In Figure 13 and Figure 14 we can see the phrase "That apple is our best apple." which will be to "**That apple is our best@(6,18,.)**" meaning that a match with the length 6 was found 18 characters in the search buffer and the next character that does not generate a match is ".".

4.2 GZIP algorithm

GZIP is a dictionary compression method based on LZ77 and Huffman codes substitutions. This method encodes a string of symbols and encodes them using a representation defined in a dictionary which can be either dynamic (adaptive) or static. The dynamic dictionary is created using strings from the previous input stream allowing the addition and deletion of new symbols as new data is processed.

"The Deflate compressed stream consists of Huffman codes for literals, lengths and distances. The literals and lengths are covered using one Huffman table and the distances use a different one. The value for literals is in the range [0:255] while the lengths are in the range [3:258]. The distances can have a maximum value of 32 Kbytes" [5].

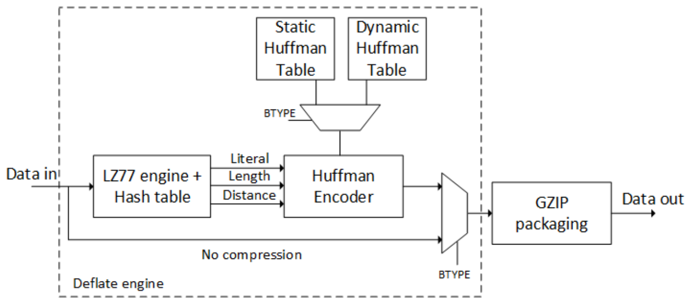
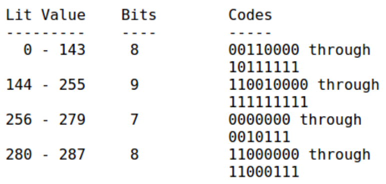


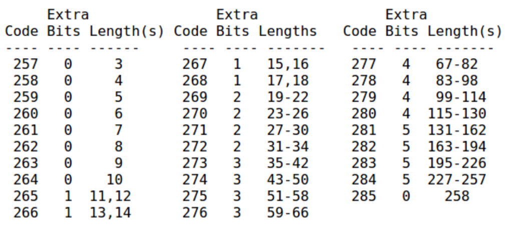
Figure 15 GZIP compression pipeline

"The codes for lengths and literals are coded in a large table with values from 0 to 287. The values from 0 to 255 represent the codes for all possible 8 bit characters while the codes in the range [257:287] are used for representing the lengths of the matches. While the Huffman codes of the literals are eight or nine bits wide, the codes used for the lengths are composed of two parts which are the extra bits of the length concatenated with the reversed length Huffman code. For example, the length 19 is encoded 00|1011000. The values for these codes can be seen in Figure 16 and Figure 17" [5].



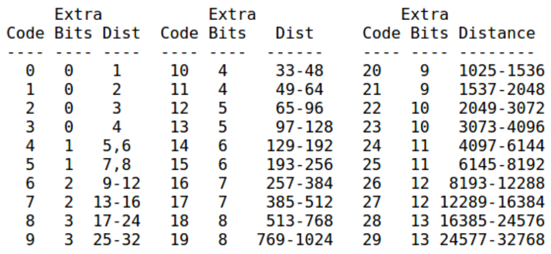
Lit Value	Bits	Codes
0 - 143	8	00110000 through 10111111
144 - 255	9	110010000 through 111111111
256 - 279	7	0000000 through 0010111
280 - 287	8	11000000 through 11000111

Figure 16 GZIP Huffman codes for literal/length [16]



Code	Extra Bits	Length(s)	Code	Extra Bits	Lengths	Code	Extra Bits	Length(s)
257	0	3	267	1	15,16	277	4	67-82
258	0	4	268	1	17,18	278	4	83-98
259	0	5	269	2	19-22	279	4	99-114
260	0	6	270	2	23-26	280	4	115-130
261	0	7	271	2	27-30	281	5	131-162
262	0	8	272	2	31-34	282	5	163-194
263	0	9	273	3	35-42	283	5	195-226
264	0	10	274	3	43-50	284	5	227-257
265	1	11,12	275	3	51-58	285	0	258
266	1	13,14	276	3	59-66			

Figure 17 GZIP Huffman codes for length [16]



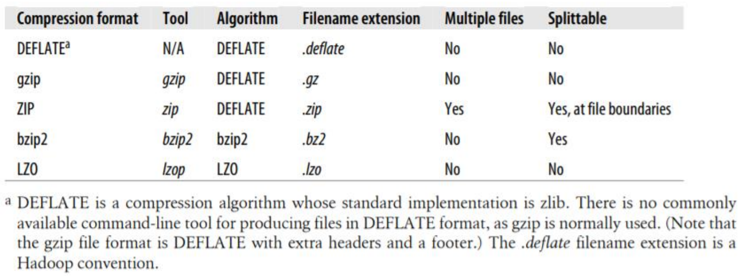
Code	Extra Bits	Dist	Code	Extra Bits	Dist	Code	Extra Bits	Distance
0	0	1	10	4	33-48	20	9	1025-1536
1	0	2	11	4	49-64	21	9	1537-2048
2	0	3	12	5	65-96	22	10	2049-3072
3	0	4	13	5	97-128	23	10	3073-4096
4	1	5,6	14	6	129-192	24	11	4097-6144
5	1	7,8	15	6	193-256	25	11	6145-8192
6	2	9-12	16	7	257-384	26	12	8193-12288
7	2	13-16	17	7	385-512	27	12	12289-16384
8	3	17-24	18	8	513-768	28	13	16385-24576
9	3	25-32	19	8	769-1024	29	13	24577-32768

Figure 18 GZIP Huffman codes for distance [16]

"The codes used to represent the distances are present in Figure 18. The maximum value of this parameter is 32768 and can be represented on a maximum of 18 bits" [5].

4.3 Data Compression in Hadoop

The compression algorithms used in Hadoop [19] can be seen in Figure 19.



Compression format	Tool	Algorithm	Filename extension	Multiple files	Splittable
DEFLATE[a]	N/A	DEFLATE	*.deflate*	No	No
gzip	*gzip*	DEFLATE	*.gz*	No	No
ZIP	*zip*	DEFLATE	*.zip*	Yes	Yes, at file boundaries
bzip2	*bzip2*	bzip2	*.bz2*	No	Yes
LZO	*lzop*	LZO	*.lzo*	No	No

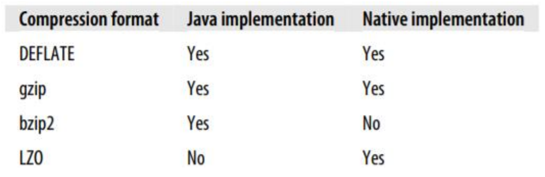
[a] DEFLATE is a compression algorithm whose standard implementation is zlib. There is no commonly available command-line tool for producing files in DEFLATE format, as gzip is normally used. (Note that the gzip file format is DEFLATE with extra headers and a footer.) The *.deflate* filename extension is a Hadoop convention.

Figure 19 Supported compression formats in Hadoop [19]

In Big Data, file compression has two major benefits:

- It speeds up the data transfers between the nodes across the network. In the same time, it increases the transfer speed from disk to memory;
- Reduces the disk space required to store the data.

“For increasing the compression performance in both speed and compression ratio, Hadoop offers the possibility to use native libraries for performing this operation. As an example, the native GZIP library brings a compression speed-up of 10% and a decompression speed increase of 2x” [19]. The algorithms with native libraries are the following:



Compression format	Java implementation	Native implementation
DEFLATE	Yes	Yes
gzip	Yes	Yes
bzip2	Yes	No
LZO	No	Yes

Figure 20 Algorithms that support native implementations in Hadoop [19]

5 Architecture and Design of a FPGA GZIP accelerator

The main candidates for hardware implementation were GZIP and BZIP. We choose the GZIP algorithm because it has better features for hardware implementation than BZIP2.

“The GZIP CORE project was designed and implemented using a ML605 [20] development board connected via PCIE in a Ubuntu 14.04 machine. The ML605 board is based on a Virtex-6 XC6VLX240T FPGA that has 240K logic cells and 14Kb of internal memory. The project also contains a software utility used to compress files in the same manner as the GZIP software and a Known Answer Test (KAT) for debugging during the design cycle. The core is connected with the Xillybus IP [21] core which can offer easy access over PCIE [22]” [5]. Later, the core was ported to a RTL version for Zynq-7020.

This is the first open source, high speed, technology independent and low area GZIP CORE [23] in the world which can be freely used for academic research.

5.1 State of the art for hardware GZIP

The latest state of the art hardware GZIP compressors are: CPU implementation @2.7Gbps (Intel) [24], FPGA implementation (IBM) @24Gbps, ASIC implementation (AHA) @20Gbps [25], HDL implementation ZipAccel-C (CAST) @40Gbps [26].

5.2 LZ77 hardware encoder

The GZIP compression software relies on the LZ77 algorithm [18].

The LZ77 hardware compression is based on [27] (DARPA [28] project) and [29].

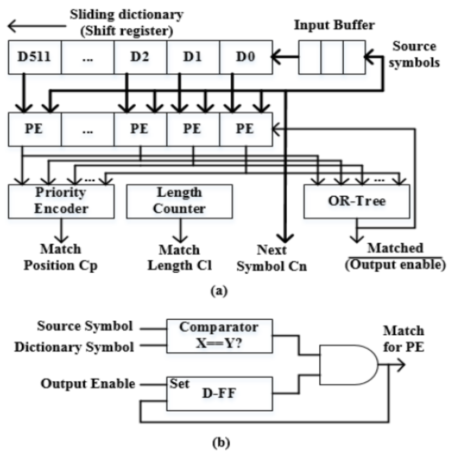


Figure 21 LZ77 systolic array: a)Encoder architecture; b) Processing Element (PE) [27]

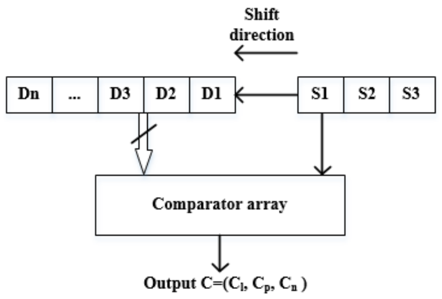


Figure 22 Simplified architecture of the LZ77 encoder [25]

“The output of the LZ77 encoder is a 3 parameters pointer $C = (C_l, C_p, C_n)$. C_l represents the length of the string match, C_p represents the location where that match was found and C_n represents the succeeding symbol in the stream” [5].

“The position of the longest match is calculated using a priority encoder. Because of the big combinational loops, several optimization techniques are implemented to enable the best results for synthesis” [5].

5.3 Architecture of the compression core

A general architecture for a GZIP core is present in Figure 23.

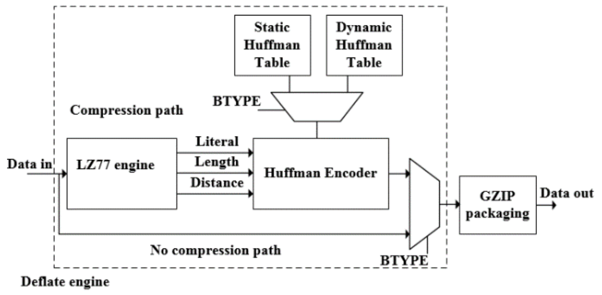


Figure 23 GZIP core general architecture [25]

Our architecture from Figure 24 supports the Stored and Fixed Huffman modes as they are better suited for a low area and high speed hardware implementation.

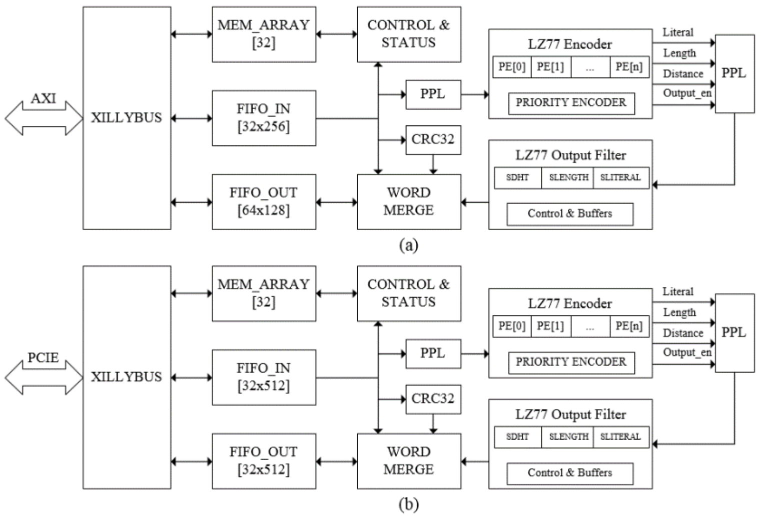


Figure 24 Detailed architecture of the designed GZIP core: a) Zynq7020 version b) Virtex6 version

The main differences between the 2 architectures from Figure 24 are described in [5] in chapter 5.3.

The architecture presented in Figure 24 is technology independent, modular and scalable and does not utilize Xilinx or Altera design blocks.

5.3.1 Xillybus

This is a hardware and software IP stack designed for transferring data between the host memory and a FPGA using DMA. This module enables easy integration with host software in Linux and also provides an embedded Linux version called Xillinux for the Zynq family [21].

5.3.2 FIFOS

The input FIFO has a size of 32x256 bits so it can buffer 1024 bytes from the DMA engine. The output FIFO has a size of 65x128 bits as the output words are 64 bits wide and another extra bit is used to flag that all bytes present in the payload are processed.

5.3.3 MEM_ARRAY

This block is an 8x32 RAM memory which stores the status and control registers for the GZIP core.

5.3.4 CRC32

This module implements the CRC-32 polynomial described in ISO 3309 which calculates the CRC of a 8 bit character C by doing the operation $P(x)\ mod\ C$ using a precomputed Look Up Table like in Figure 25.

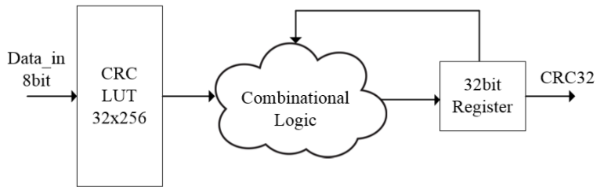


Figure 25 CRC32 design with LUT

5.3.5 LZ77 Encoder

This block returns the value of the match pointers calculated between the symbols already inserted in the dictionary and the ones present in the look-ahead buffer.

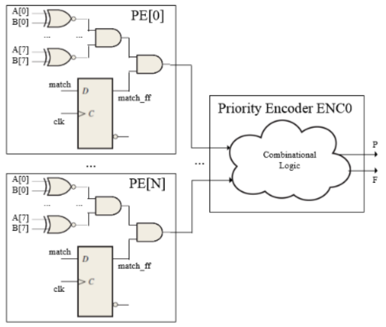


Figure 26 Initial combinational logic tree in the LZ77 encoder [30]

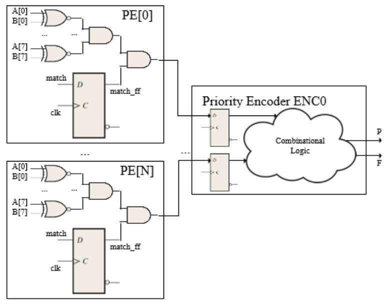


Figure 27 LZ77 first improvement for synthesis

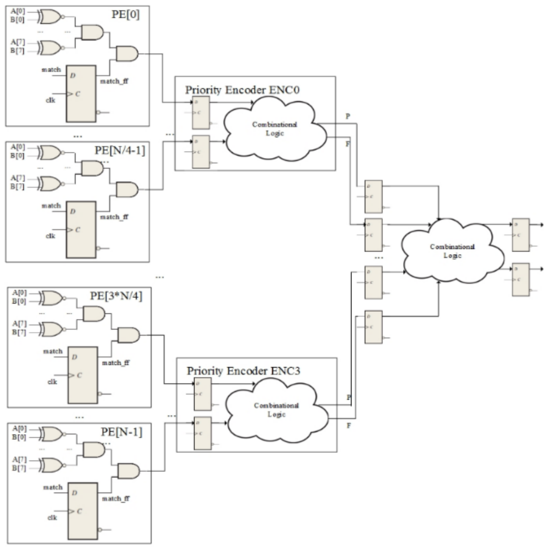


Figure 28 LZ77 second improvement for synthesis [30]

Because this module represents the "critical path" of the design, we had to split the priority encoder into smaller ones and add pipelining stages between them in order to obtain better synthesis results.

5.3.6 LZ77 Filter

The purpose of this module is to align the output of the LZ77 encoder in such a manner that they are compliant with the Deflate format described in RFC 1951.

5.3.7 SDHT – Static Distance Huffman Tree

This module is used to compute the distance values detected by the LZ77 encoder.

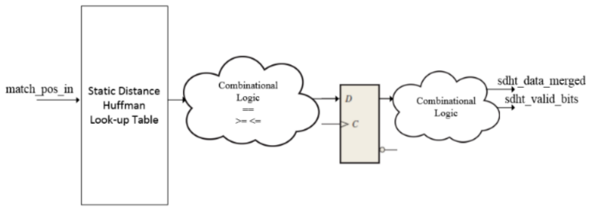


Figure 29 SDHT architecture

5.3.8 SLITERAL – Static Literal Huffman Tree

This module is used to transform from ASCII values, which can be from 0 to 255, to Huffman encoded values.

5.3.9 SLENGHT – Static Literal Huffman Tree

This module is used to encode with Huffman codes the value of the length pointer.

5.3.10 WORD MERGE

This role of the WORD MERGE module is to concatenate input lengths between 1 and 64 bits and to pack them into 64 bits chunks.

5.3.11 GZIP TOP

This module is the top level module which comprises all the modules mentioned above with the glue logic necessary to interconnect them. The most important part of this module is the state machine (Figure 30.) which is used to read the data coming from the software utilitary through Xillybus IP.

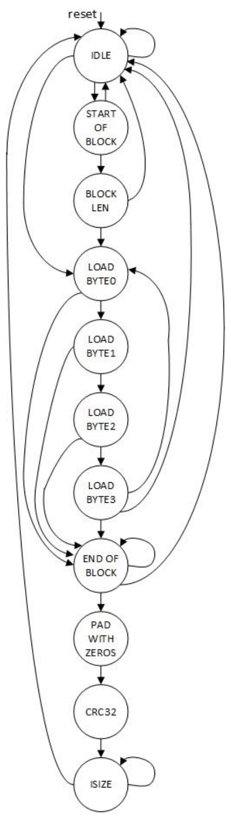


Figure 30 GZIP CORE state machine [30]

5.4 Operation mode and software design

5.4.1 GZIP core register space

The GZIP core has several registers **RST_REG, BTYPE_REG**, **DEBUG_REG1…16, DEV_ID.**

Address	Name	Field Name							
	Bit position	7	6	5	4	3	2	1	0
0	RST_REG	x	x	x	x	x	x	x	RST_N R/W
1	BTYPE_REG	x	x	x	x	x	SEL_ENDNS R/W	BTYPE[1:0] R/W	
2	DEBUG_REG1	x	x	x	x	x	GZIP_DONE RO	BTYPE_ERR RO	BSIZE_ERR RO
3	DEBUG_REG2	ISIZE[31:24] RO							
4	DEBUG_REG3	ISIZE[23:16] RO							
5	DEBUG_REG4	ISIZE[15: 8] RO							
6	DEBUG_REG5	ISIZE[7 : 0] RO							
7	DEBUG_REG6	CRC32[31:24] RO							
8	DEBUG_REG7	CRC32[23:16] RO							
9	DEBUG_REG8	CRC32[15: 8] RO							
10	DEBUG_REG9	CRC32[7 : 0] RO							
11	DEBUG_REG10	BLOCK_SIZE[23:16] RO							
12	DEBUG_REG11	BLOCK_SIZE[15: 8] RO							
13	DEBUG_REG12	BLOCK_SIZE[7: 0] RO							
14	DEBUG_REG13	OUTPUT_SIZE[31:24] RO							
15	DEBUG_REG14	OUTPUT_SIZE[23:16] RO							
16	DEBUG_REG15	OUTPUT_SIZE[15: 7] RO							
17	DEBUG_REG16	OUTPUT_SIZE[7: 0] RO							
18	DEV_ID	DEV_ID[7:0] RO							
Legend:	x = unimplemented								
	R/W = read/write								
	RO = read only								

Table 3 GZIP core register address space

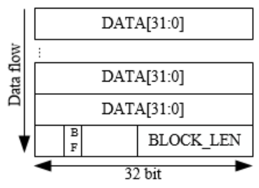
5.4.2 GZIP core compression software

The core is operated by using a software routine written in C which packs the basic data transfer functions provided by the Xillybus IP Linux driver. The software splits the input file into smaller blocks, sends the data to the FPGA using the write pipe and then the output is expected on the read pipe. The software also writes the GZIP file header, thus the archive can be decompressed by any GZIP compliant software.

A command word as in Figure 31 (a) is used to control the GZIP core state machine.

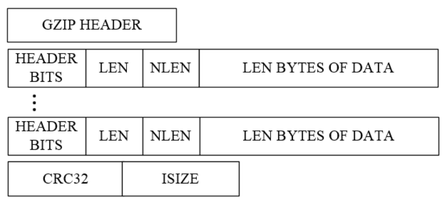
Byte	3								2								1								0							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Subfield	x	x	x	x	x	x	x	BF	x	x	x	x	x	x	x	x	BLOCK_LEN[15:0]															

(a)

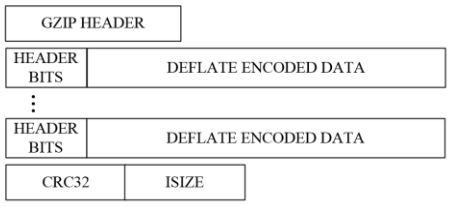


(b)

Figure 31 Data stream from CPU to FPGAs



(a)



(b)

Figure 32 Structures of the compressed streams

In Figure 32 (a) and (b) we can see the structure of the compressed data streams created by the software and GZIP core in the STORED and FIXED_HUFFMAN modes.

5.5 Experimental results

5.5.1 GZIP core synthesis results on Virtex6

The Deflate core was synthesized in standalone mode and the synthesis area consumption for both chips is present in Table 4.

Dictionary Depth	Xilinx				Altera			
	Number of Slice Registers [%]	*Number of Slice LUTs [%]*	*Max clock frequency [MHz]*	*Max throughput [Mbps]*	*Number of Registers [%]*	*Number of ALMs [%]*	*Max clock frequency [MHz]*	*Max throughput [Mbps]*
512	1.5	2.5	168	1344	0.56	1.6	230	1840
1024	2.5	4.5	165	1320	1.14	2	210	1680
2048	5.5	9	150	1200	2.25	5	160	1280
4096	11	19	148	1184	4.45	10	100	800

Table 4 Deflate core synthesis on Xilinx and Altera FPGAs [25]

Dictionary Depth	Number of Slice Registers [%]	Number of Slice LUTs [%]	Max clock frequency [MHz]	Max throughput [Mbps]
512	2	3	164	1312
1024	3	5	167	1336
2048	6	10	149	1192
4096	13	20	144	1152

Table 5 GZIP core synthesys on Xilinx Virtex6 [25]

Table 5 contains the synthesis results for the GZIP core from Figure 24 (a) on the Virtex6 FPGA. The extra logic slightly affects the performance of the core meaning that the pipelining successfully limits the delays on the critical paths.

5.5.2 GZIP core performance measurements on Virtex6

The selected dictionary size for the GZIP core is 512 and the look-ahead buffer has a size of 66. The GZIP core clock frequency is 100MHz, synthesized along the Xillybus IP, and programmed in the ML605 board.

Calgary Corpus Results	Intel i7 @ 2.8 GHz	FPGA GZIP @ 100MHz
Average Speed [MBps]	2.84	6.5
CR	3.3	2.1
Max speed [MBps]	5.5	28
MAX CR	9	7.2

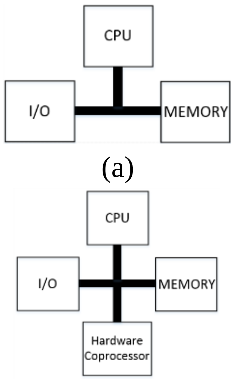
Table 6 GZIP profiling using the Calgary corpus on Virtex6

5.5.3 Comparison with the state of the art compressors

"While the compressors from Section 5.1 are faster than our core, we have to take in consideration that they are industrial IPs developed by large companies and the performance is measured using very expensive FPGA that have superior performance. Our GZIP core has a small area and can achieve good performance on low-cost and energy-efficient FPGAs also" [5].

6 Evaluation of a Hadoop ZedBoard cluster with FPGA GZIP acceleration

The GZIP core was integrated in a Hadoop heterogeneous cluster based on ZedBoard development platforms. The difference between classical data processing using the CPU and high-speed processing using specialized computing units is shown in Figure 33 (a) and (b). The specialized processing element usually has a lower clock speed and a higher throughput due to the optimized hardware architecture.



(b)

Figure 33 Hardware coprocessor usage diagram

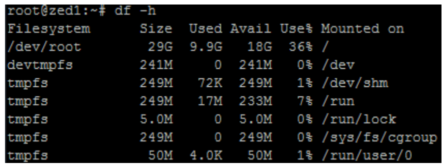
ARM servers are a research field that explores the usage of low-power CPUs combined with hardware accelerators [31], [32].

6.1 State of the art for Heterogeneous ZedBoard clusters

“Heterogenous Hadoop cluster for low power processing have been previously studied in [33] to accelerate FIR filter processing and in [34] to accelerate graph-oriented applications. In [33] the FPGA acceleration brings a 20% speed improvement compared with the ARM processor working independently while in [34] the FPGA acceleration makes the cluster 1.2x times faster for graph processing operations” [5].

6.2 Xillinux prerequisites for a ZedBoard Datanode

The steps for setting up each slave node are described in [5]. A very important step is to resize the SD card as in Figure 34.



```
root@zed1:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        29G  9.9G   18G  36% /
devtmpfs        241M     0  241M   0% /dev
tmpfs           249M   72K  249M   1% /dev/shm
tmpfs           249M   17M  233M   7% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs            50M  4.0K   50M   1% /run/user/0
```

Figure 34 SD card after resize

6.3 GZIP core synthesis results on Zynq7020

The maxim clock speed the core can synthesize in the Zynq7020 FPGA fabric is 184MHz for a dictionary 512 bytes deep and maximum value for the look-ahead buffer of 258 characters.

FPGA Resource	**GZIP core**	**GZIP core + Xillybus IP**
LUT	9%	21%
LUTRAM	2%	6%
FLIPFLOPS	4%	12%
BRAM	3%	1%

Table 7 GZIP core synthesis on Zynq7020

6.4 GZIP core performance measurements on Zynq7020

For profiling the performance of the GZIP core, we used several popular corpuses for profiling data compression performance: Silesia, Calgary, Canterbury, Lukas 3D, Protein.

Corpus name	**Silesia**		**Calgary**		**Canterbury**		**Lukas 3D**		**Protein**	
Compession processor	***ARM***	***FPGA***	***ARM***	***FPGA***	***ARM***	***FPGA***	***ARM***	***FPGA***	***ARM***	***FPGA***
Average speed (MBps)	0.59	4.68	0.37	2.2	0.28	2.15	0.74	3.47	0.8	3.9
CR	3.7	1.78	3.3	2.14	3.6	1.75	1.88	1.36	1.72	1.04
Max speed (MBps)	1.18	4.87	0.56	4.4	0.38	4.1	0.85	4.21	0.85	4.79
Max CR	10.48	4.24	9.09	6.61	9.09	6.61	1.95	1.39	1.74	1.05

Table 8 GZIP core profiling on Zynq7020 [35]

Table 8 shows the compression speed and compression ratio using some of the best-known corpuses used to profile data compression efficiency. We can see that the compression ratio is smaller up to 2.47x times than that of the GZIP software while the latter has a dictionary 64 times larger. The compression speed is up to 5.6x times higher than the speed of the ARM CPU.

The power and energy consumption of the GZIP core was measured using a MCP39F501 board from Microchip and created the setup from Figure 35.

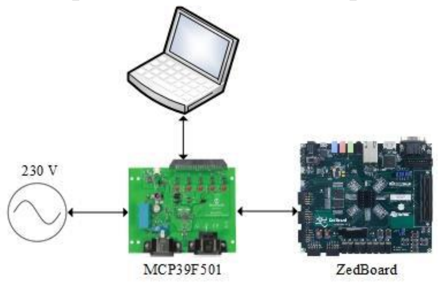


Figure 35 ZedBoard power measurement setup [35]

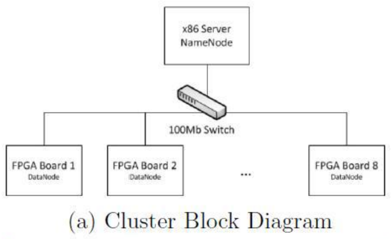
The average energy and power consumption of the ZedBoard when performing hardware and software compression can be seen in Table 9.

File Size [MB]	ARM + FPGA			ARM			Energy ARM / Energy (ARM + FPGA)
	***Power* [W]**	***Time* [s]**	***Energy* [J]**	***Power* [W]**	***Time* [s]**	***Energy* [J]**	
100 MB	5.25	12.5	65.6	5.05	69.30	349.9	5.33
160 MB	5.25	25	131.25	5.05	148.20	748.4	5.69
450MB	5.26	62	326.1	5.05	290	1464.5	4.49
950 MB	5.28	143	755	5.06	602	3046	4.03

Table 10 GZIP core power and energy profiling [35]

6.5 FPGA GZIP core integration in Hadoop

In Figure 36 (a) and (b) we have the diagram of the ZedBoard cluster.



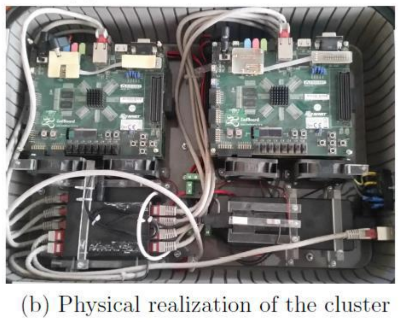


Figure 36 Hadoop ZedBoard cluster architecture [35]

The native GZIP compression libraries from Hadoop were replaced with a custom library that used our GZIP core for compressing data.

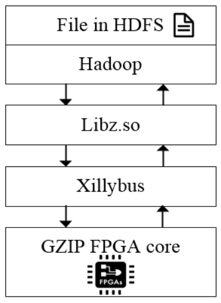


Figure 37 GZIP compression data flow in Hadoop

6.6 Cluster benchmarking

For profiling the performance of the ZedBoard cluster we the popular Hadoop benchmarks Wordcount and Terasort.

6.6.1 Wordcount results

We cans see in Figure 38 the data compression process add a 12% processing overhead to the total duration of the benchmark. Table 11 shows an energy consumption improvement of 44% when the hardware compression core is used.

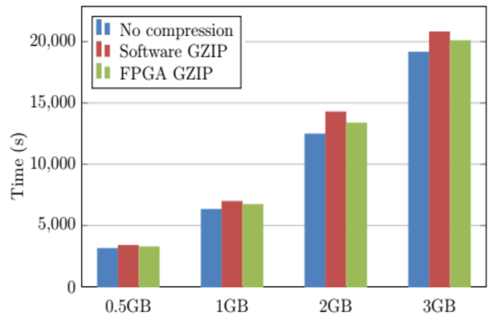


Figure 38 Wordcount benchmark results [35]

Data Size [GB]	No GZIP [J]	Soft GZIP (normalised)	GZIP core (normalised)	Improvement between FPGA and soft GZIP %
0.5	137992	1.077	1.04	47.73
1	278990	1.103	1.063	38.59
2	550334	1.146	1.074	49.25
3	845413	1.088	1.051	41.61

Table 11 Cluster energy consumption for Wordcount [35]

6.6.2 Terasort results

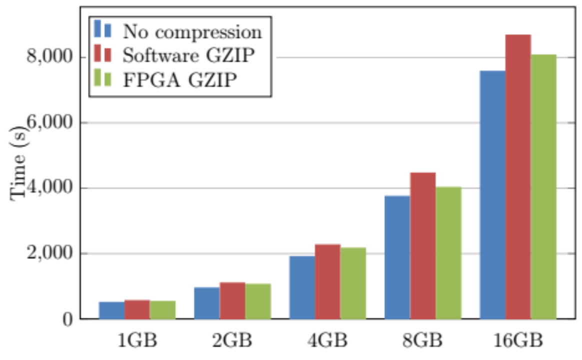


Figure 39 Terasort benchmark results [35]

“In Figure 39 we can see that the compression adds and computation overhead of 13% in the total duration of the Terasort benchmark while in Table 12 it is shown that the average energy consumption improvement has an average of 42%” [5].

Data Size (GB)	**No GZIP (J)**	**Soft GZIP (norm)**	**GZIP core (norm)**	**Improvement between FPGA and soft GZIP %**
1	22276	1.106	1.053	49.73
2	41636	1.163	1.163	25.50
4	84024	1.192	1.192	28.08
8	165705	1.191	1.191	54.07
16	334638	1.147	1.147	53.60

Table 12 Cluster energy consumption for Terasort [35]

6.6.3 Comparison with the state of the art Heterogeneous ZedBoard clusters

“The other ZedBoard clusters are accelerating other compute-intensive operations using the FPGA fabric, so our experiments are original and can be used as a baseline for later and more efficient studies about data compression in Big Data heterogeneous clusters.” [5].

7 Conclusions

7.1 Experimental results interpretation

All 3 initial objective of the thesis were reached during the research stages:

1. **Build an ARM based Hadoop cluster and measure the speed and energy efficiency of the cluster using standard Hadoop benchmarks.**
2. **Implement a GZIP FPGA accelerator.**
3. **Accelerate data compression in a Hadoop heterogeneous cluster.**

The processing speed of the Hadoop cluster is strongly dependent on the data storage speed, network fabric and processing resources, as CPU and RAM.

“The values from both benchmarks done on the Hadoop ZedBoard cluster suggest that the FPGA hardware compression brings an energy improvement up to 54% during the compression stages” [5].

“Our core can reach a maximum bandwidth of 1.84Gbps on a StratixV device and reaches the maximum 100MB speed using large files on a Linux machine. The core can be 18x faster than an Intel i7 CPU running at 28x higher clock frequency. The core is also up to 5.6x times faster than an ARM CPU running at a frequency 6.7 times higher” [5].

“From the energy saving perspective, the GZIP core is up to 5.6x times more energy-efficient than the ARM CPU processing the same payload. If the compression is applied in the Hadoop Map and Reduce stages, the core is almost 2 times more energy-efficient than the ARM processor in the same stages” [5].

7.2 Original contributions of this thesis

“The key contributions of this thesis are:

1. **The first open-source, technology independent, high-speed GZIP FPGA compression core and LZ77 engine. The core is publicly available at** [23]**. The repository contains the Verilog sources, the module level tests, the system level tests and the compression software used for operating the core via Xillybus. The used programming and scripting languages are: Verilog, C, Bash and Perl.**
2. **An optimized architecture for a high speed and low area FPGA compression core. The core is up to 18x times faster than an Intel i7 CPU clocked at a frequency 28x times bigger. Compared to the ARM CPU, the core is up to 8x times faster while the ARM core frequency is 6.7x times higher. The core uses less than 10% of the resources of a Zynq-7020 FPGA.**
3. **The first ZedBoard Hadoop heterogeneous cluster with GZIP FPGA acceleration. The speed and energy consumption improvements are measured. The cluster is up to 2x times more energy efficient when performing compression using the GZIP core.**
4. **We have proven that heterogeneous ARM FPGA devices could be a solution for future energy-efficient data processing machines”** [5]**.**

7.3 List of published articles

“The results obtained during the research stages were published in four articles:

1. **"EVALUATION OF A LOW-POWER HADOOP CLUSTER" U.P.B. Scientific Bulletin, p. 12, 2017** [14]**.**
2. **"FPGA systolic array GZIP compressor," in International Conference on Electronics, Computers and Artificial Intelligence, Targoviste, Romania, 2017, (ISI Web of Science indexed)** [25]**.**
3. **"Hadoop ZedBoard cluster with GZIP compression FPGA acceleration," in 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Targoviste, Romania, 2019, (ISI Web of Science indexed)** [35]**.**
4. **"Optimizing GZIP compression accelerator for Zynq FPGAs," in International Semiconductor Conference (CAS), Sinaia, Romania, 2019** [30]**.”** [5]

7.4 Future development

It would be interesting to add clock constraints for better synthesis results and to implement the parallel GZIP compression algorithm described in the PIGZ algorithm [36].

8 References (partial)

[1] GYGABYTE, "https://www.gigabyte.com/ARM-Server/Cavium-ThunderX2," GIGABYTE. [Online]. [Accessed 2019].

[2] S. L. J. L. Olle Svanfeldt-Winter, "Cost and Energy Reduction Evaluation for ARM Based Web Servers," in *9'th International Conference on Dependable, Autonomic and Secure Computing*, Sydney, NSW, Australia, 2011.

[3] L. M. G. G. Rafael Vidal Aroca, "Towards green data centers: A comparison of x86 and ARM," *Journal of Parallel and Distributed Computing,* vol. 72, no. 12, pp. 1770-1780, 2012.

[4] S. J. C. R. P. B. S. J. N. S. O. M. S. James T. Cox, "Iridis-pi: A low-cost, compact demonstration cluster," *Cluster Computing* , vol. 17, no. 2, 2013.

[5] O. Plugariu, "Hardware Offloading for Energy-Efficient Distributed Computing," University "Politehnica" of Bucharest, Bucharest, 2019.

[6] U. E. P. Agency, "www.energystar.gov," [Online]. Available: http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf. [Accessed 2017].

[7] A. Shimoni, "https://hackernoon.com/a-gentle-introduction-to-hardware-accelerated-data-processing-81ac79c2105," [Online]. [Accessed 2018].

[8] M. Z. T. C. Z. S. Y. M. B. Y. Qianru Zhang, "Recent advances in convolutional neural network acceleration," *Neurocomputing,* 2019.

[9] A. Putnam, "FPGAs in the Datacenter: Combining the Worlds of Hardware and Software Development," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, Banff, Alberta, Canada, 2017.

[10] G. B. P. C. G. S. Jeremy Fowers, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," in *FPGA '12 Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, Monterey, California, USA, 2012.

[11] J. S. D. S. A. M. S. K. D. M. Eriko Nurvitadhi, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *26th International Conference on Field Programmable Logic and Applications (FPL)*, Lausanne, Switzerland, 2016.

[12] A. Hadoop, 2017. [Online]. Available: https://hadoop.apache.org/.

[13] R. Menon, ""Introducing Hadoop, part II"," 2016. [Online]. Available: http://www.rohitmenon.com/index.php/.

[14] A. C. a. L. P. Ovidiu PLUGARIU, "EVALUATION OF A LOW-POWER HADOOP CLUSTER," *U.P.B. Scientific Bulletin,* p. 12, 2017.

[15] "Creating a text corpus from wikipedia.," [Online]. Available: http://trulymadlywordly.blogspot.ro/2011/03/creating-text-corpus-from-wikipedia.html.. [Accessed 07 04 2016].

[16] D. Solomon, Data compression: The Complete Refference - 4th edition, Springer, 2006.

[17] G. C. M. Ribeiro, "Data Compression Algorithms in FPGAs," Tecnicio Lisboa, Lisbon, Portugal, 2017.

[18] A. L. Jacob Ziv, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory* , 1977.

[19] T. White, Hadoop The Definitive Guide, Sebastopol, CA 95472: O'Reilly Media, Inc., 2009.

[20] X. Inc., "www.xilinx.com," 2012. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-v6-ml605-g.html. [Accessed 2 2014].

[21] X. Ltd., "http://xillybus.com/," [Online]. [Accessed 6 2013].

[22] Wikipedia, "en.wikipedia.org," [Online]. Available: https://en.wikipedia.org/wiki/PCI_Express. [Accessed 7 2013].

[23] L. P. Ovidiu Plugariu, "gitlab.dcae.pub.ro," [Online]. Available: https://gitlab.dcae.pub.ro/research/FPGA-GZip. [Accessed 8 2017].

[24] A. H. D. S. Mohamed S. Abdelfattah, "Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL," in *Proceedings of the International Workshop on OpenCL*, Bristol, United Kingdom, 2014.

[25] A. D. G. L. P. Ovidiu Plugariu, "FPGA systolic array GZIP compressor," in *International Conference on Electronics, Computers and Artificial Intelligence* , Targoviste, Romania, 2017.

[26] CAST Inc., "http://www.cast-inc.com/news/2017-10-17-cast-introduces-gzip-accelerator-through-new-intel-fpga-data-center-acceleration-ecosystem," CAST, 2017. [Online]. [Accessed 5 2019].

[27] N. S. a. E. J. M. Wei-Je Huang, "A Reliable LZ Data Compressor on Reconfigurable Coprocessors," in *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines* , Napa Valley, CA, USA, 2000.

[28] DARPA, "https://www.darpa.mil/," [Online]. [Accessed 2015].

[29] A. E. S. A. H. K. Mohamed A. Abd El Ghany, "Design and Implementation of FPGA- based Systolic Array for LZ Data Compression," in *IEEE International Symposium on Circuits and Systems*, New Orleans, LA, USA, 2007.

[30] O. Plugariu, "Optimizing GZIP compression accelerator for Zynq FPGAs," in *International Semiconductor Conference (CAS)*, Sinaia, Romania, 2019.

[31] Y. M. T. Bogdan Marius Tudor, "On understanding the energy consumption of ARM-based multicore servers," in *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*, Pittsburgh, PA, USA , 2013.

[32] M. P. J. H. M. P. S. K. M. v. d. B. T. K. W. C. René Griessl, "A Scalable Server Architecture for Next-Generation Heterogeneous Compute Clusters," in *IEEE International Conference on Embedded and Ubiquitous Computing*, Milano, Italy, 2014.

[33] P. C. Zhongduo Lin, "ZCluster: A Zynq-based Hadoop cluster," in *International Conference on Field-Programmable Technology (FPT)*, Kyoto, Japan, 2013.

[34] N. K. Pradeep Moorthy, "Zedwulf: Power-Performance Tradeoffs of a 32-Node Zynq SoC Cluster," in *IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, Vancouver, BC, Canada, 2015.

[35] L. P. R. P. R. H. Ovidiu PLUGARIU, "Hadoop ZedBoard cluster with GZIP compression FPGA acceleration," in *11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Targoviste, Romania, 2019.

References (partial)

[36] A. Mark, "https://zlib.net/pigz/," [Online]. Available: https://zlib.net/pigz/. [Accessed 12 4 2019].

[37] Wikipedia, "www.wikipedia.org," [Online]. Available: https://en.wikipedia.org/wiki/Data_center#Energy_efficiency. [Accessed 2016].

[38] Racktivity, "www.racktivity.com/," 2017. [Online]. Available: http://www.racktivity.com/uptime-maximization-opex-/.

[39] AVNET, "www.zedboard.org," 2017. [Online]. Available: http://zedboard.org/product/zedboard.

[40] X. Inc.. [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/operating-systems.html. [Accessed 2016].

[41] A. Hadoop, 2016. [Online]. Available: http://mirrors.hostingromania.ro/apache.org/hadoop/common/.

[42] SSH, 2016. [Online]. Available: https://www.ssh.com/ssh/#sec-The-SSH-protocol.

[43] F. Ludovic, "Compression Algorithms," Ecole Polytechnique Federale de Lausanne, 2010.

[44] D. J. W. M. Burrows, "A block-sorting lossless data compression algorithm," Digital Systems Research Center, Palo Alto, 1994.

[45] M. F. Oberhumer, "http://www.oberhumer.com/," [Online]. Available: http://www.oberhumer.com/opensource/lzo/#abstract. [Accessed 6 2016].

[46] L. P. Deutsch, "DEFLATE Compressed Data Format Specification version 1.3," 1996. [Online]. Available: https://www.w3.org/Graphics/PNG/RFC-1951. [Accessed 5 2016].

[47] Y. Nekritch, "Decoding of canonical Huffman codes with look-up tables," in *Proceedings DCC 2000. Data Compression Conference*, Snowbird, UT, USA, USA, 2000.

[48] J. Seward, "bzip2," 2010. [Online]. Available: https://en.wikipedia.org/wiki/Bzip2. [Accessed 4 4 2016].

[49] A. Ltd. [Online]. Available: https://community.arm.com/developer/ip-products/system/b/soc-design-blog/posts/introduction-to-axi-protocol-understanding-the-axi-interface. [Accessed 2014].

[50] Technopedia. [Online]. Available: https://www.techopedia.com/definition/2767/direct-memory-access-dma. [Accessed 2015].

[51] X. Ltd.. [Online]. Available: http://xillybus.com/xillinux. [Accessed 2 2017].

[52] Y. K. J. V. A. K. P. Rauschert, "Very fast GZIP compression by means of content addressable memories," in *IEEE Region 10 Conference TENCON* , Chiang Mai, Thailand, 2004.

[53] H. L. Z. W. J. T. C. L. K. S. Jian Ouyang, "FPGA implementation of GZIP compression and decompression for IDC services," in *International Conference on Field-Programmable Technology*, Beijing, China, 2010.

[54] C.-Y. L. Ren-Yang Yang, "High-Throughput Data Compressor Designs Using Content Addressable Memory," in *Proceedings of IEEE International Symposium on Circuits and Systems*, London, UK, 1994.

[55] C. F. U. R. C. P. Virgilio Zuñiga Grajeda, "Parallel Hardware/Software Architecture for the BWT and LZ77 Lossless Data," National Institute for Astrophysics, Optics and Electronics, Luis Enrique Erro, Puebla, Mexico, 2006.

[56] G. G. F. L. Ameer M. S. Abdelhadi, "Deep and narrow binary content-addressable memories using FPGA-based BRAMs," International Conference on Field-Programmable Technology (FPT), Shanghai, China, 2014.

[57] Wikipedia, "https://en.wikipedia.org/wiki/OpenCL," [Online]. [Accessed 2017].

[58] A. H. D. S. Mohamed S. Abdelfattah, "Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL," in *Proceedings of the International Workshop on OpenCL 2013 & 2014*, Bristol, United Kingdom, 2014.

[59] J. Pasternack, "https://engineering.linkedin.com/blog/2019/02/migz-for-compression-and-decompression," LinkedIn, 2019. [Online]. Available: https://engineering.linkedin.com/blog/2019/02/migz-for-compression-and-decompression. [Accessed 3 5 2019].

[60] A. M. C. E. S. C. D. C. K. C. J. D. H. E. J. F. K. C. J. D. H. E. Andrew Putnam, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," *IEEE Micro,* vol. 35, no. 3, pp. 10-22, 2015.

[61] A. G. R. H. K. Yanpei Chen, "To compress or not to compress - compute vs. IO tradeoffs for mapreduce energy efficiency," in *Proceedings of the first ACM SIGCOMM workshop on Green networking*, New Delhi, India, 2010.

[62] A. Allain, "https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html," [Online]. [Accessed 2018].

[63] Xilinx, "https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842223/U-boot," [Online]. [Accessed 2017].

[64] Xillinx, "https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842279/Build+Device+Tree+Blob#BuildDeviceTreeBlob-Whatisdevicetree?," [Online]. [Accessed 2017].

[65] Putty, "https://www.putty.org/," [Online]. [Accessed 2015].

[66] A. W. Services, "https://aws.amazon.com/ec2/instance-types/f1/," Amazon. [Online]. [Accessed 2017].

[67] X. Inc., "https://www.xilinx.com/products/boards-and-kits/alveo.html," Xilinx. [Online]. [Accessed 2019].

[68] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, no. 3, 1948.

[69] I. W. J. G. C. Timothy C. Bell, "Modeling for Text Compression," *ACM Computing Surveys,* vol. 21, no. 4, pp. 557-591, 1989.

[70] PCISIG, "https://pcisig.com/specifications," [Online].

[71] X. Wiki, "https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/21430418/Zynq-7000+AP+SoC+SATA+part+1+Ready+to+Run+Design+Example+Setup," Xilinx Inc.. [Online]. [Accessed 3 2019].

[72] Techcrunch, "https://techcrunch.com/2019/05/14/zombieload-flaw-intel-processors/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=CmwaF2qP_hd7LFdb0aUu0Q," [Online]. [Accessed 14 5 2019].

[73] M. Adler, "https://github.com/madler/infgen/blob/master/infgen.c," 3 1 2017. [Online]. [Accessed 7 2017].

[74] A. table, "http://www.asciitable.com/," [Online].

[75] N. Katayoun, M. Maria, G. M. Ali, S. Avesta and H. Houman, "Energy-efficient acceleration of big data analytics applications using FPGAs," in *IEEE International Conference on Big Data (Big Data)*, Santa Clara, CA, USA, 2015.

[76] O. Plugariu, "Hardware Offloading for Energy-Efficient Distributed Computing," University "Politehnica" of Bucharest, Bucharest, 2019.