METHODOLOGY FOR WORKLOAD ANALYSIS AND SCHEDULING ON HETEROGENEOUS EMBEDDED SYSTEMS

by

GAREFALAKIS EMMANOUIL

TECHNOLOGICAL EDUCATION INSTITUTE PIRAEUS

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF INFORMATICS ENGINEERING

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

2015

Approved by:

Supervisor
Dr KORNAROS GEORGIOS

Committee
Dr PAPADAKIS NIKOLAOS
Dr STRATAKIS DIMITRIOS

Acknowledgements

I especially want to thank my supervisor Kornaros G.
for his guidance and help that made this thesis possible.

To my wife Maria, for being supportive and encouraging.

To my son Giorgos, for the time I spent away from him.

Manos Garefalakis

Contents

Acknowledgements I
Contents III
List of Figures IV
List of Tables VI
Abstract 7
Chapter 1 - Introduction 8
Motivation – Thesis Objectives 8
Thesis Organization 10
Chapter 2 - Heterogeneous System Architecture (HSA) 11
Heterogeneous Systems 12
CPU – General Purpose GPU 12
Embedded Systems 13
OpenCL 15
HSA Intermediate Language (HSAIL) 16
Chapter 3 - Experimental Evaluation 19
Experimental setup 19
Target Platform 19
Benchmark Description 24
Evaluation metrics 27
Experimental Results 28
Measurements Graphs 28
Chapter 4 - Workload time prediction 51
Worst Case Execution Time 51
Regression analysis 52
Coefficient of determination 53
Modeling kernel for WCET 53
Chapter 5 - Kernels Concurrent Scheduling 69
Scheduling Simulation 70
Conclusions 72
Future Work 73
Bibliography 74

List of Figures

Figure 2.1: A generic embedded system [41] 15
Figure 2. 2: NDRange, Work-Group, Work-Item [40] 18
Figure 2.3: Software Stack HSA Foundation [40] 18

Figure 3.1: Avnet ZedBoard - Zynq-7000 19
Figure 3.2: Xilinx Zynq-7000 SoC Architecture [14, 17] 20
Figure 3.3: FPGA programmable logic [17] 22
Figure 3.4: MicroBlaze Architecture [20] 23
Figure 3.5: MicroBlaze Soft-Core Block Diagram [18] 23
Figure 3.6: Execution times required for solving the equation $aX^3+bX^2+cX+d=0$ with constants a,b,c,d within the ranges $-10<a<10$, $-10<b<10$, $-5<c<15$, $-3<d<3$ 29
Figure 3.7: Execution times for computing the SQRT of Unsigned Integer(Unsigned Integer range within 0 - 4FEBB399) 30
Figure 3.8: Execution Times for the conversion of Degrees to Radians Degrees Range within 0^o-500^o 31
Figure 3.9: Execution Times for the computation of a stock price for a range of days within 10 – 365 32
Figure 3.10: Execution Times for creating a CRC code for a range of text size within 10-660 bytes... 33
Figure 3. 11: Execution Times for finding 5 shortest paths in a Graph of adjacent Nodes within the range of 5 - 50 33
Figure 3. 12: Execution Times to find 5 – 50 shortest paths in a Graph of 50 adjacent Nodes 34
Figure 3. 13: Execution Times to execute the hash function for a range of bytes within 25- 9925 bytes 34
Figure 3. 14: Execution Times for computing FFT & IFFT values for a range within 2 – 512..... 35
Figure 3. 15 : Execution Times for applying median filter on images with image size with 16 - 900 pixels 37
Figure 3. 16: Sample of Image Denoise Kernel tests 37
Figure 3. 17: Execution Times for applying Edge Detection on Images with image size within 16 – 196 pixels 38
Figure 3.18: Different aspect of the 3D Figure 3.12 with Region 1 and 2 40
Figure 3.19: Different aspect of the 3D Figure 3.12 40
Figure 3. 20: Execution Times for searching strings within 50-1000 chars in strings within 50-1000 chars (Caching Disabled) 40
Figure 3. 21: Different aspect of the 3D Figure 3.15 with Region 1 and 2 41
Figure 3.22: Different aspect of the 3D Figure 3.15 41
Figure 3.23 : Execution Times for K-means Clustering for Objects with Coordinates 2 Number of Objects within 60-120 Number of Clusters 10 – 60 (Caching Enabled)..... 43
Figure 3.24 : Different aspect of 3D Figure 3.23 43
Figure 3. 25 : Different aspect of 3D Figure 23 43
Figure 3. 26 : Execution Times for K-means Clustering, for Objects with Coordinates 2, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Disabled) 44
Figure 3. 27 : Different aspect of 3D Figure 3.26 44
Figure 3. 28 : Different aspect of 3D Figure 3.26 44
Figure 3. 29 : Execution Times for K-means Clustering for Objects with Coordinates 3 Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Enabled)..... 45

Figure 3. 30 : Different aspect of 3D Figure 3.29 45
Figure 3. 31 : Different aspect of 3D Figure 3.29 45
Figure 3. 32 : Execution Times for K-means Clustering for, Objects with Coordinates 3, Number of Objects within 60-120,Number of Clusters 10 – 60 (Caching Disabled) 46
Figure 3. 33 : Different aspect of 3D Figure 3.32 46
Figure 3. 34 : Different aspect of 3D Figure 3.32 47
Figure 3. 35 : Execution Times for K-means Clustering for Objects with Coordinates 4, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Enabled) 47
Figure 3. 36 : Different aspect of 3D Figure 3.35 47
Figure 3. 37 : Different aspect of 3D Figure 3.35 47
Figure 3. 38 : Execution Times for K-means Clustering for Objects with Coordinates 4, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Disabled) 48
Figure 3. 39 : Different aspect of 3D Figure 3.38 48
Figure 3. 40 : Different aspect of 3D Figure 3.38 48
Figure 3.41 : Execution Times for K-means Clustering for Objects with Coordinates 5, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Enabled) 49
Figure 3.42 : Different aspect of 3D Figure 3.41 49
Figure 3. 43 : Different aspect of 3D Figure 3.41 49
Figure 3. 44 : Execution Times for K-means Clustering for Objects with Coordinates 5, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Disabled) 50
Figure 3.45 : Different aspect of 3D Figure 3.45 50
Figure 3. 46 : Different aspect of 3D Figure 3.44 50

Figure 4.1 : Execution times required for solving the equation $aX^3+bX^2+cX+d=0$ with constants a,b,c,d within the ranges $-10<a<10$, $-10<b<10$, $-5<c<15$, $-3<d<3$ 54
Figure 4.2 : Execution times for computing the SQRT of Unsigned Integer 54
Figure 4.3 : Execution Times for the conversion of Degrees to Radians Degrees Range within 55
Figure 4.4 : Execution Times for the computation of a stock price for a range of days within 10 – 365 56
Figure 4.5 : Execution Times for creating a CRC code for a range of text size within 10-660 bytes.... 56
Figure 4.6 : Execution Times for finding 5 shortest paths in a Graph of adjacent Nodes within the range of 5 - 50 57
Figure 4.7 : Execution Times to find 5 – 50 shortest paths in a Graph of 50 adjacent Nodes 57
Figure 4.8: Execution Times to execute the hash function for a range of bytes within 25- 9925 bytes 58
Figure 4.9 : Execution Times for computing FFT & IFFT values for a range within 2 – 512..... 59
Figure 4.10 : WCET for computing FFT values for a range within 2 – 512 (Caching Enabled)..... 59
Figure 4. 11 : WCET for computing FFT values for a range within 2 – 512 (Caching Disabled)..... 59
Figure 4. 12 : WCET for computing IFFT values for a range within 2 – 512 (Caching Disabled) 60
Figure 4.13 : Execution Times for applying median filter on images with image size with 16 - 900 pixels 61
Figure 4.14 : Execution Times for applying Edge Detection on Images with image size within 16 – 196 pixels 61
Figure 4. 15 : Execution Times for searching strings within 50-1000 chars in strings within 50-1000 chars (Caching Enabled) 62
Figure 4. 16 : Execution Times for searching strings within 50-1000 chars in strings within 50-1000 chars (Caching Disabled) 62
Figure 4. 17 : Execution Times for K-means Clustering for Objects with Coordinates 2 Number of Objects within 60-120 Number of Clusters 10 – 60 (Caching Enabled)..... 63

Figure 4. 18 : Execution Times for K-means Clustering for Objects with Coordinates 2, Number of Objects within 60-120, Number of Clusters 10 – 60(Caching Disabled) 64
Figure 4. 19 : Execution Times for K-means Clustering for, Objects with Coordinates 3,Number of Objects within 60-120,Number of Clusters 10 – 60 (Caching Enabled)..... 65
Figure 4. 20 : Execution Times for K-means Clustering for Objects with Coordinates 3, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Disabled) 65
Figure 4. 21: Execution Times for K-means Clustering for Objects with Coordinates 4, Number of Objects within 60-120, Number of Clusters 10 – 60(Caching Enabled) 66
Figure 4. 22 : Execution Times for K-means Clustering for Objects with Coordinates 4,Number of Objects within 60-120,Number of Clusters 10 – 60(Caching Disabled) 67
Figure 4. 23 : Execution Times for K-means Clustering for Objects with Coordinates 5, Number of Objects within 60-120, Number of Clusters 10 – 60(Caching Enabled) 67
Figure 4. 24 : Execution Times for K-means Clustering for Objects with Coordinates 5 Number of Objects within 60-120 Number of Clusters 10 – 60(Caching Disabled) 68

Figure 5. 1: Schedule for 7 tasks on 2 cpus..... 70
Figure 5. 2 : Schedule for 7 tasks on 4 cpus..... 71
Figure 5. 3 : Schedule for 7 tasks on 2 cpus with power optimization 71
Figure 5. 4 : Schedule for 7 tasks on 4 cpus (With Power Optimization)..... 72

List of Tables

Table 2.1 : Common Design Requirements for Embedded Systems [41] 15

Table 3.1: Zynq I/O controllers..... 21
Table 3.2: MicroBlaze Processor v8.40.b Performance Levels (v14.4 XPS) 24
Table 3.3: List of Selected Kernels..... 27
Table 3.4: Sample Results of Cubic Solving Tests..... 28
Table 3.5 : Sample Result of Unsigned SQRT tests 29
Table 3.6 : Sample of Deg2Rad(x) Tests..... 30
Table 3.7 : Sample results of the BlackScoles Tests..... 31
Table 3.8 : Sample Results of the CRC32 tests 32
Table 3. 9: Sample Results of SHA-1 tests..... 34
Table 3. 10 : Samples of the FFT tests..... 36
Table 3. 11 : Sample of Image Edge Detection 38
Table 3. 12:Execution Times for searching strings within 50-1000 chars in strings within 50-1000 chars(Caching Enabled) 39

Table 5. 1 : Independent Task Simulation 70
Table 5. 2: Independent Tasks Simulation with power consumption optimization 71

Abstract

The shift towards multicore technologies is offering a great potential of computational power for scientific and industrial applications. However, great challenges to software development arise. Performance gains for data intensive and compute intensive applications can be achieved by exploiting coarse-grained and fine-grained parallelism on all system levels and improved scalability with respect to the increasing core counts as Butchy et al. state [4]. Reconfigurable hardware has received increasing attention in the past decade due to its capability of being adaptable, short design time and low cost. Instead of using field-programmable gate arrays (FPGAs) just as application specific integrated circuit (ASIC) replacements, designers can combine reconfigurable hardware with general purpose processors in a co-design system, providing a flexible and powerful means of implementing computational applications [35].

In this work the first steps for the implementation of such a co-design system are provided. The methodology that is proposed consists of:

a) selection of kernels or applications that will be modified and executed by the processing elements
b) analysis and modeling of the kernels in order to perform execution time prediction depended on data input
c) multiprocessors scheduling of independent tasks minimizing mean execution time

The final project leads to a many cores System on Chip (SoC) with the ability to execute data intensive and compute intensive applications, taking into account optimal performance and power consumption.

Keywords: FPGA, Kernels, Heterogeneous System Architecture, Workload Execution Time prediction, Scheduling Multiprocessors

Chapter 1 - Introduction

In the last decades, the evolution of embedded systems progresses immensely. Embedded systems have a very important role in the eras of technology and industry. This fact emerges the great need of implementation of methods that assist and accelerate the design of embedded systems and the evaluation of their performance from the first stages of design. It is also well known that processing units, such as Graphics Processing Units (GPUs) and accelerators can execute specialized processes faster than General Purpose Processors (CPUs). But the transfer of the processing data from the hosts to the accelerators and the transfer back of computation results to the hosts is a cost effective procedure in time and memory resources, especially in embedded device with constrained resources.

This thesis is about the design of a methodology that aims to optimize performance of constrained resources in an embedded device with heterogeneous architecture (HSA). The main purpose is the optimal performance and reduced power consumption in heterogeneous system architecture (HSA). The hardware platform that will be used contains an ARM Host (CPU) and accelerators in which the performance of the system will be logged in order to achieve the best implementation. The software used will be open source code derived from various domains such as image processing (filter implementation, edge detection, etc.), digital signal processing (*k*-means, fft, inverse fft, etc.) and models from the econometrics e.g. Black Scholes Merton.

Finally the thesis closes with the implementation and simulation of a task scheduler based on the partition approximation algorithm applied on many cores architecture.

Motivation – Thesis Objectives

The challenges regarding todays multicore technology and advancements that allow integration of multiple accelerators in embedded systems was the motivation for this thesis. The work of Zhong & He [25] with the idea that kernels are being executed from GP/GPU in slices (small pieces) and scheduled in a smart way was the motivation of this thesis.

In the paper of Zhong & He [25] the implementation is performed using the CUDA framework, which clearly obtained a high degree of acceptance within the high-performance computing community; However, CUDA is a single vendor that works only with NVidia hardware [2].

The main idea is such as kernels are executed on GP/GPUs they could also be executed on specific accelerators, soft-cores with lower power consumption and lower cost which can be

included in a System on Chip in contrast to the GPUs that are expensive devices and have also other constrains.

It could be achieved to run concurrent kernels on specific accelerators with different resources and possibilities, using message passing technology or networks on chip. The kernels could be preloaded on the accelerators and a program that manages the system to send input data and get the results of the computations from each accelerator. The accelerators will be dedicated and configured in an optimal way for specific tasks in the heterogeneous system.

In order to complete the whole project some stages must be completed. The first stage is to find programs or kernels from different domains that can be executed from the accelerators. The code should be modified in such a way to receive input data and send the results to a system manager. This is the first objective of this thesis.

The system manager has to be able to schedule the kernels, independent or dependent tasks, reducing mean finishing time of the workload [26]. For scheduling reasons it is important to know the execution time of each kernel. The execution times of the kernels are dependent on the input data set, which means that it is needed to model the execution times of each kernel for a range of input data set. So, it is required to create a function which will include the kernel models and returns the execution times for specific input data set. In order to create a model for each kernel it is necessary to conduct tests with the kernels for a range of input data set and log the execution times. This is the second objective of this thesis.

The last objective of this thesis but not last stage of the project is to implement a scheduling algorithm for many cores in order to be executed from the system manager.

Finally, after the three first stages, the most important stage is to combine all work together, something that is beyond the limits of this thesis and is part of the future work.

Thesis Organization

This thesis is organized as follows:

- In the section 2 titled “Heterogeneous System Architecture”, the basic concepts of HSA and a hardware-aware on multicore processors and accelerators are presented.
- In the section 3 titled “Experimental Setup”, the list of selected kernels and the procedure followed in order to perform the tests and logging of execution times of each kernel is presented.
- In the section 4 titled “Workload time prediction”, the methodology followed in order to model each kernel for predicting execution time dependent on different data inputs is explained in details.
- In the Section 5 titled “Kernels Concurrent Scheduling”, a proposed algorithm for multiprocessors scheduling and a simulation of the algorithm is described in order to show the effectiveness of the algorithm.
- The thesis completes with a brief presentation of what was done in the conclusions and closes with the future work. In the Future work the problems that researchers have to tackle and the next steps that should be done are presented.

Chapter 2 - Heterogeneous System Architecture (HSA)

According to Su [1], heterogeneous system architecture represents a new era in computer architecture. Its primary goal is that computer system designers must tightly integrate different processing elements on a platform into one evolved central processor while providing a way to software designers that does not require fundamentals changes. In other words heterogeneous in computing refers to systems that use more than one kind of processors like CPUs, GPUs, DSPs and other programmable accelerators tightly integrated into a single System on Chip (SoC).

Kyriazis [3] in his technical report concludes that HSA is a unified computing framework. It provides a single address space accessible to both CPU and GPU in order to avoid data copying. It also provides user-space queuing for minimizing communication overhead and finally provides preemptive context switching for better quality of service across all computing elements in the system. In HSA CPUs and GPUs are unified into a single system with common computing concepts and allows the developer to solve a greater variety of complex problems more easily.

Brookwood [2] explains in a simple case paradigm how HSA works efficiently. For example a platform like a smartphone, needs a CPU to run operating system tasks and various applications (e.g. web browsing, video games), additionally there is a need of handling the Graphics computations, which would increase the computational resources of the CPU that would consequently increase the power consumption. Modern GPUs are specialized to handle such computations more efficiently by accelerating the computations and by reducing the power consumption. A smartphone must also handle Digital Signal Processing (DSPs) tasks in real time, like converting the voice into binary stream and transmit it to the next cell phone tower using different radio protocols and vice versa receive binary stream, reconstructing it to voice that can be heard from the phone speaker. Additionally, users want to interact naturally with their systems; they want their products to recognize faces, track eye movements, respond to touches, voice and gestures. These tasks are accomplished more efficiently by contemporary DSPs Chips than a general purpose CPU.

Each of these programming elements, CPUs, GPUs and DSPs are evolved separately the last decades and designers assembled them in the systems they designed. Nowadays the increased transistors budget permits them to place the CPU, the GPU and the DSP discrete elements onto a single System on Chip (SoC). This physical integration enables the creation of smaller

devices, reduces the cost and saves power due to the on-chip communication, rather than communications on a board.

Heterogeneous Systems

CPU – General Purpose GPU

According to Brookwood [2] the use of discrete GPUs to accelerate compute intensive applications has gained popularity since it was first introduced by NVidia in the year 2006. A great Number of supercomputers in the Top500 list rely on AMD and NVidia GPUs to deliver breathtaking performance. A total of ninety (90) systems on the list are using accelerator technology, up from seventy five (75) on November 2014. More than fifty (52) of these use Nvidia chips, four (4) use ATI Radeon and there are now thirty five (35) systems with Intel MIC technology, the Xeon Phi. Four (4) systems use a combination of Nvidia and Intel (Xeon Phi) accelerators (http://top500.org/lists/2015/06/ last accessed 7/10/2015).

Additionally Brookwood highlights the big difference between discrete GPUs than integrated GPUs SoC. Discrete GPUs have a raw performance advantage due to the fact that they contain more floating point hardware and utilize dedicated high speed memory to store the parallel programs and data. Despite the fact that with discrete GPUs there is the need of data transfer between main memory and GPU dedicated memory using the PCIe bus whose bandwidth is limited, the time and power consumption and the complicated programming model has a high acceptance from the high performance community. On the contrary integrated GPUs possess less raw floating point processors performance and can operate on data wherever it resides within the system's memory and there is no need to move data.

Varbanescu et al. [36] describes the General Purpose GPU programming models, as a combination of coarse-grained and fine-grained parallelism. The host CPU sends the data-parallel kernels as large collections of threads on the GPU. The GPUs are used for data parallel workloads where thousands of threads can compute concurrently.

They also describe the programming models of NVidia CUDA, AMD Brook, PGI Fortran and Pathscale ENZO and they conclude that almost all are very close to the architecture. All models approach parallelization by identifying and offloading the kernels to be accelerated. They differ only to the way the offload is performed. CUDA and Brook require from the users to write code for this offload while PGI and ENZO work by inserting pragmas from available sequential code. Furthermore, the models handle the massive parallelism in the kernels in different ways. Data distribution is simplified but it is not optimized. Mapping and scheduling

are left to the hardware. Finally the effective use of the accelerator memory hierarchy should be handled by the programmer.

Embedded Systems

By searching the literature we can find many definitions of what is an embedded system. Barr [41] in his book defines that an "*embedded system is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a specific function"*. Kamal [42] in his book defines an embedded system as *" a system that has embedded software and computer-hardware, which makes it a system dedicated for an application(s) or specific part of an application or product or part of larger system"* and also lists a number of other definitions.

Sriram & Bhattacharyya [44] define that embedded systems are computers that are not first and foremost computers. They appear in different systems like telecommunications, automobiles, aircraft, electronics, toys, trains, security systems, weapons systems, printers, routers, copiers, manufacturing systems, etc. Someone who is technically active person today probably interacts regularly with more embedded systems than conventional computers. This is a relatively recent phenomenon. It is not long time ago since automobiles depended on finely tuned mechanical systems for the timing of ignition and its synchronization with other actions. Also, many electronic devices, especially in telecommunications, were finely tuned analog circuits.

Barr [41] highlights that in some cases, it would be possible even to build integrated chip (IC) that does not contain a microprocessor/s and software. This is possible by replacing the combination with a custom integrated circuit (IC) that performs the same functions in hardware. But, flexibility is lost when a design is hard-coded in this way. It is easier and cheaper to change the software than to redesign a piece of custom hardware.

History and Future of Embedded Systems

As it concerns the History and the Future of embedded systems Barr [41] says that the first systems of that kind could not possibly have appeared before the year of 1971. It was the year Intel introduced the world's first microprocessor. The chip 4004 was designed for use in a line of business calculators produced by Busicom, a Japanese company. Intel proposed a general-purpose chip that could be used throughout the entire line of calculators. Intel's idea was that the software would give to each calculator its unique set of features.

Many of the electronic devices that surround us are embedded systems or include embedded systems. Some examples are microwave ovens, TVs, HiFi System, fax machines, laser printers, card readers, phones, cars and so on. It seems inevitable that the number of embedded systems will continue to increase in high rates. Clearly, persons with the skills and desire to design advanced or next generation embedded systems will be in great demand for the next years [41].

Real-Time Systems

One category of embedded systems worth's mentioning at this point is the real-time systems. A real-time system is required to complete its work and serve its services on a timely basis. In other words, real-time systems have strict timing requirements that they must meet. Examples of real-time systems include digital control, signal processing, communicating systems, etc. All these systems provide us with services that are related to high reliability and safety [45].

Real-time and embedded systems are gaining more and more importance in our society. Recognizing the importance of these systems, the National Science Foundation has recently established a research program dedicated to embedded systems. The European Union (EU), European countries and Asian countries have also established many research programs in real-time and embedded systems. Therefore, we can look forward to many important and exciting results being developed in this area [45].

Variations on Embedded Systems

Unlike software designed for general-purpose computers, embedded software cannot usually run on other embedded systems without significant modification, mainly because of the incredible variety in the underlying hardware. The hardware in each embedded system is tailored specifically to the application, in order to keep system costs low. The common hardware in embedded systems is a processor or more than one, memories ROM and RAM can in some cases, when there is no need of big amount of memory, be included on the same chip or otherwise in external memory chips. Additionally there are inputs, connections to sensors, buttons, etc. and outputs, connections to LEDs, LCD displays, etc. Figure 2.1 depicts a general block diagram of an embedded system.

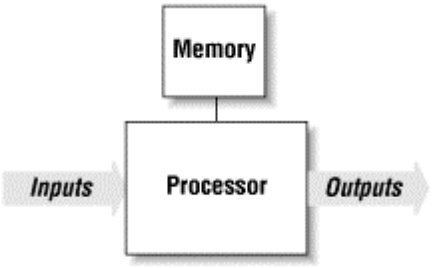


Figure 2.1: A generic embedded system [41]

The rest of embedded hardware is usually unique and they meet many competing design criteria. Each system has different set of requirements, some or all of which can affect the arrangements and tradeoffs made during the development of the product.

Barr [41] lists and analyses the list of possible constraints under which embedded hardware designers work. Some constrains are the production cost, the processing power, the amount of memory, the development cost, the Expected lifetime, the Number of units and the Reliability.

To these general requirements there are also detailed functional requirements of the system itself. These are things that give the embedded system its unique identity.

The Table 2.1 illustrates the range of possible values for each of the previously mentioned design requirements. These are only estimates and should not be taken too seriously. In some cases the criteria are linked and are dependent with each other [41].

Criterion	**Low**	**Medium**	**High**
Processor	4-bit or 8-bit	16-bit	32-bit or 64-bit
Memory	< 16 KB	64 KB to 1 MB	> 1 MB
Development cost	< $100,000	$100,000 to $1,000,000	> $1,000,000
Production cost	< $10	$10 to $1,000	> $1,000
Number of units	< 100	100-10,000	> 10,000
Expected lifetime	days, weeks, or months	years	Decades
Reliability	may occasionally fail	must work reliably	must be fail-proof

Table 2.1 : Common Design Requirements for Embedded Systems [41]

OpenCL

The OpenCL (Open Computing Language) is an open standard for general purpose parallel programming across CPUs, GPUs and other processors, giving to software-developers portable and efficient access to the power of these heterogeneous processing platforms. It is a framework for parallel programming and includes a language, API, libraries and a runtime

system to support software development. With OpenCL for example a programmer can code general purpose programs that are executed on GPUs without the need to map their algorithms onto a 3D graphics API such as DirectX or OpenGL.

Furthermore, the OpenCL offers a common API for program execution on systems composed of different types of computational devices such as many-core CPUs, GPUs or other accelerators [38]. It supports data-parallel and task-parallel programming models, it utilizes a subset of ISO C99 with extensions for parallelism, it defines consistent numerical requirements based on IEEE 754, it defines a configuration profile for handheld and embedded devices, and it efficiently interoperates with OpenGL, OpenGL ES and other graphics APIs.

The OpenCL programming interfaces assume heterogeneity between the host and all the attached devices. Additionally, the programming interfaces include functions that [39]

- enumerate available target devices like CPUs, GPUs, and various accelerators
- manage the target devices context
- manage memory allocations
- perform host-devices memory transfers
- compile the OpenCL programs and kernel functions that the devices will execute
- launch kernels on the target devices
- query execution progress
- check for errors

With OpenCL kernel portability and accuracy across a variety of hardware is guaranteed, but it is not guarantee that a particular kernel will achieve peak performance on different processors architectures; the hardware's features might make some programming strategies more suitable for particular platforms than for others [39].

The strongest point of OpenCL is its portability by construction, due to the fact that using the *common platform model* as a middleware. The OpenCL back-end targets one machine type only and it is the responsibility of the vendors to provide the appropriate OpenCL drivers. That's why it is good motivation for all processor vendors to develop high-performing OpenCL APIs for their platforms [36].

HSA Intermediate Language (HSAIL)

Kyriazis [3] in his technical review mentions that Heterogeneous System Architecture (HSA) provides a unified view of the computing elements. The HSA allows a programmer to develop applications that integrate CPUs (*latency compute units - LCUs*) with GPUs (*throughput*

compute units - TCUs) without any gaps and benefit from the best attributes of each computing elements. The GPUs have passed in recent years from pure graphics accelerators to general purpose parallel processors, supported by standard APIs and tools like DirectCompute and OpenCL™. Those APIs are a very good start, but many obstacles remain for the creation of an environment that permits the GPU to be used with such an ease as the CPU for common programming tasks. The HSA removes those obstacles, and allows the programmer to take the advantages of the parallel processor in the GPU as a peer or co-processor to the traditional multithreaded CPU.

On the one hand the LCU in a HSA environment is a generalization of a CPU that supports apart from its native instruction set (ISA) and the HSA intermediate language (HSAIL) instruction set. On the other hand the TCU is the generalization of a GPU that supports only the HSA intermediate language (HSAIL) instruction set, which allows them to perform very efficient parallel execution of tasks or threads.

The idea is that the compiler for a high-level language (e.g. OpenCL, C++ AMP, Java, etc.) will generate HSAIL and the HSA driver, called finalizer, will generate the binary code using just-in-time compilation. The idea of a pseudo-ISA has been used, long time ago, in many previous portable software technologies like the Direct3D bytecode and the Java bytecode. The HSAIL is low-leveled enough to uncover many details of the hardware and has been carefully designed in such a way that the conversion from HSAIL to binary code is done very quickly ("A Deep Dive on HSA", http://www.anandtech.com/show/7677/amd-kaveri-review-a8-7600-a10-7850k/6 last accessed 15/10/2015). The finalizer, the converter to binary code, is typically lightweight and may be executed at install time, compile time, or program execution time, all depends on the choices made by the platform implementation.

A question that arises is "*Why Virtual ISA and Not the Real ISA?*". Rubin gives answers to this question:

- ISA Gains performance
- Better time to market (because hardware is finished faster)
- Loses performance (cannot use every hardware trick)
- No legacy boat anchor
- Real ISA means one vendor or one chip family
- Hardware bugs can be fixed via software
- Code works on old and new machines
- Allows hardware innovation (under the table)
- Features not in HSAIL are not exposed, and are hard to access

(http://www.slideshare.net/hsafoundation/hsail-final-11junepptx last accessed 14/10/2015)

In Figure 2.2 we can see the parallel model that the HSAIL complies with.

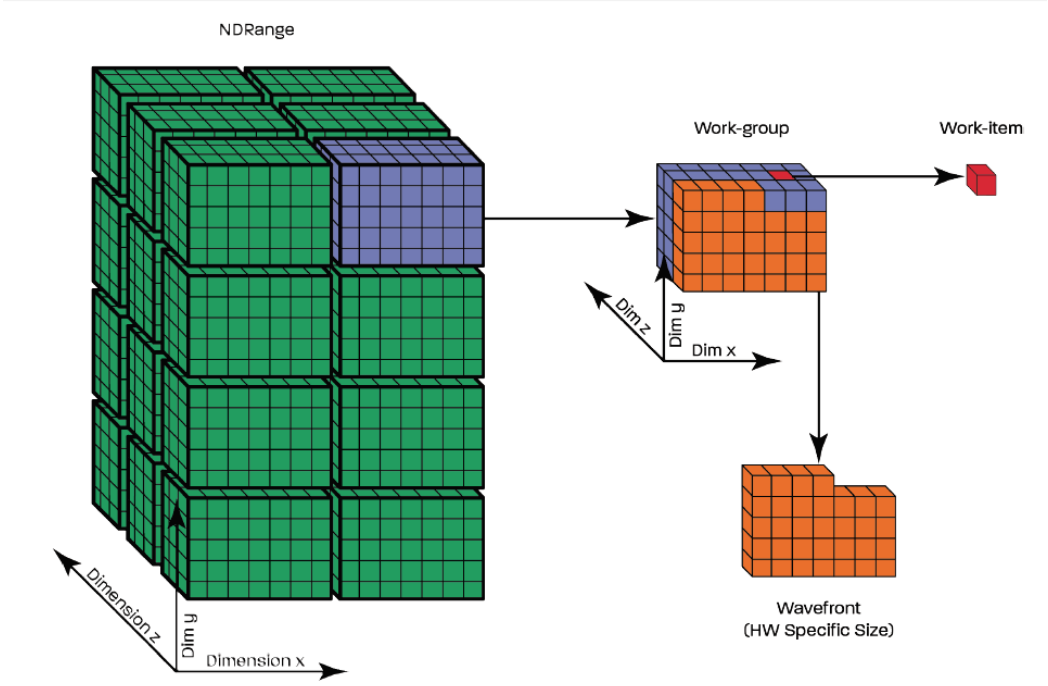


Figure 2.2: NDRange, Work-Group, Work-Item [40]

In Figure 2.3 we can see the Software Task of HSA. The HSAIL is lower than OpenCL that's why it not another OpenCL but it extends the features of OpenCL.

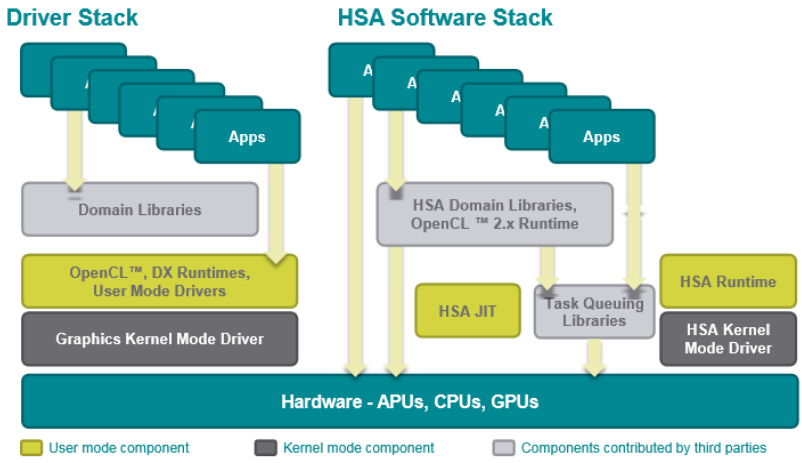


Figure 2.3: Software Stack HSA Foundation [40]

Chapter 3 - Experimental Evaluation

Experimental setup

Target Platform

The Xilinx's Zynq-7000 SoC is divided into two sectors: the processing system and the programmable logic. The processing system Figure 3.2 can be viewed as a microcontroller featuring an ARM Cortex-A9 embedded processor. The programmable logic is equivalent to a low-cost field programmable gate array (FPGA), which contains programmable hardware Figure 3.3. A block diagram of the Zynq-7000 can be seen in Figure 3.2. Experiments for this work were conducted on an Avnet ZedBoard [14], see Figure 3.1, a development board which contains a Zynq-7000 [17].

Figure 3.1: Avnet ZedBoard - Zynq-7000

Zynq-7000 Processing System

In Figure 3.2 there is the block diagram of the Zynq-7000 processing system. The processing system consists of the following discrete elements:

- an application processor unit (APU)
- memory controllers
- peripheral controllers

The APU is based on an ARM Cortex-A9 dual-core embedded processor. The fastest clock speed is up to 1GHz. The default frequency clock speed of the ARM Cortex-A9 is 667 MHz. For the experimental setup the ARM Cortex-A9 operated at its default frequency. Each processing core features its own vector and floating point unit, a memory management unit

(MMU) and 64KB of Level 1 (L1) cache memory (32KB instruction, 32KB data). A memory of 512KB of L2 cache is shared between the cores, along 256KB of on-chip SRAM. The APU also includes an eight-channel Direct Memory Access (DMA) controller, system control registers, a global interrupt controller and various timers. Outside the APU, the Zynq processing system contains huge number of interconnects, interfaces and peripheral controllers. An external memory interface allows both the processing system and the programmable logic to be connected to external DDR memory. In the case of the ZedBoard, 512MB of DDR3 memory is available. Interfaces to NOR, NAND, and QSPI external ash memory are included for nonvolatile storage options. Table 3.1 shows the peripheral controllers available on the Board. External storage is available via USB ash drives, SD cards or SPI-connected ash. Gigabit Ethernet provides high speed communication. Lower-speed communication is available via USB, UART, CAN, SPI, and I2C. A GPIO module allows control over LEDs, switches, and various I/O pins. These I/O modules are memory-mapped and multiplexed so as to conserve pins when certain modules are unused. Multiple general-purpose and high-performance interconnects allow interfacing of the processing system with the programmable logic. An accelerator coherency port (ACP) provides direct access to L2 cache and on-chip memory for modules in the programmable logic. A cryptography block and an analog to digital converter (ADC) are located within the programmable logic but are available for the processing system's use [14, 17].

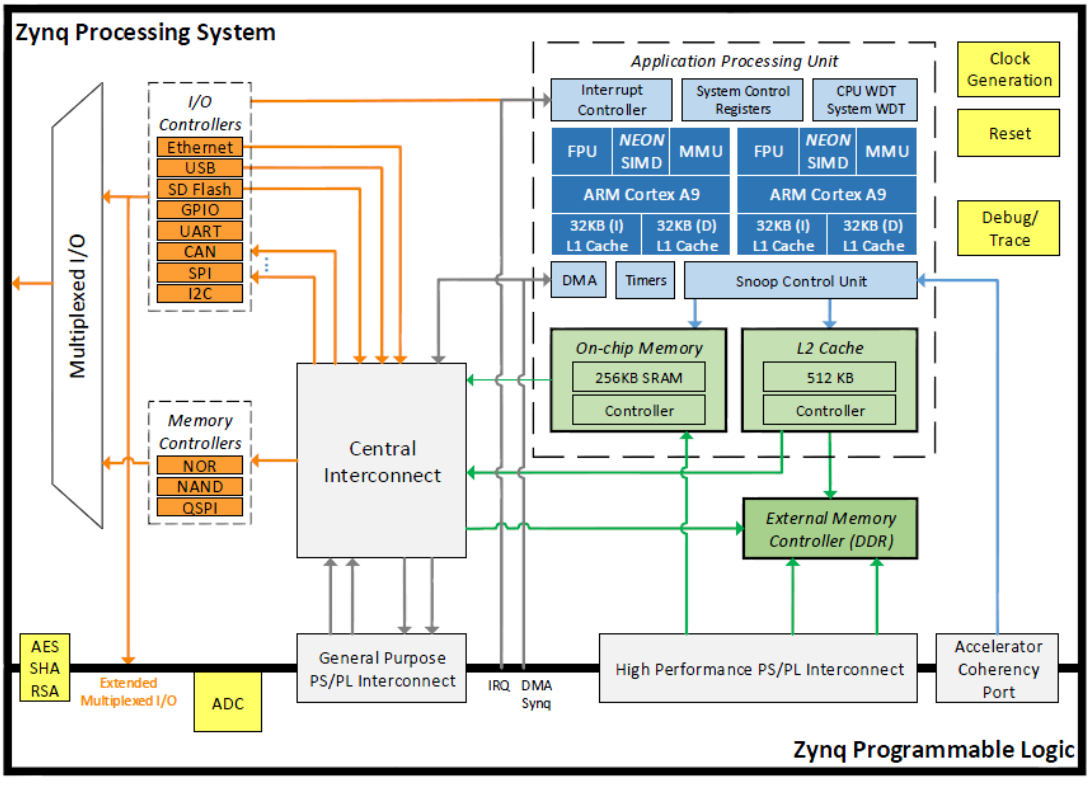


Figure 3.2: Xilinx Zynq-7000 SoC Architecture [14, 17]

I/O Type	Number Available	Example Use
SD flash	2	External storage
General Purpose I/O	1	LED, switch, etc.
Gigabit Ethernet	2	Network access, high-speed communication
UART	2	Communication
CAN	2	Communication
USB	2	External memory, communication
SPI	2	External memory, low-speed communication
I2C	2	External memory, low-speed communication

Table 3.1: Zynq I/O controllers

Programmable Logic

Depending on the package chosen, Zynq's programmable logic is equivalent to either a low cost (Xilinx Artix-7) or high-performance (Xilinx Kintex-7) field programmable gate array (FPGA). An FPGA is a semiconductor device that has no predetermined function. Unlike an application-specific integrated circuit (ASIC), whose operation is fixed during the manufacturing process, an FPGA's function can be programmed repeatedly after manufacturing. FPGAs are an ideal platform for prototyping potential designs or for distributing a low volume of ICs. Typically, a function is implemented in hardware in order to improve performance or power consumption over a similar software implementation. Because an FPGA is a programmable hardware, a function can be duplicated many times with each instance running in parallel, thus improving performance. While not commonly considered low power devices, an FPGA implementation of a highly-parallel function may draw considerably less power than the same operation implemented on a PC. Performance and power are the conventional reasons for FPGA usage [17].

A generic means of creating combinational hardware is by using lookup tables (LUT). LUTs simply store the appropriate output corresponding to each potential input of a function Flip-ops (FF) are available at the outputs of LUTs as a storage element. LUTs and FF are grouped into slices, and slices are combined into configurable logic blocks (CLB). A typical FPGA architecture is shown in Figure 3.3. A sizable number of CLBs are included, along with block RAM (BRAM) for use when a large amount of memory is required. Digital Signal Processing (DSP) slices allow for faster processing of common arithmetic operations. I/O blocks (IOBs) interface the design with external pins. These components connect via a programmable interconnect. Connections may usually be made at the intersection of any two routing paths, permitting cross-chip connections and communication. Each CLB is comprised of two slices,

for a total of four LUTs and eight FFs. LUTs can be configured as 6-input/1-output or 5-input/2-output. BRAMs are dual-port and contain 36Kb of memory; each can be addressed as a single unit or as two independent 18Kb BRAMs. DSP slices include a 25-bit pre-adder, 48-bit adder and accumulator, 25x18 signed multiplier [15].

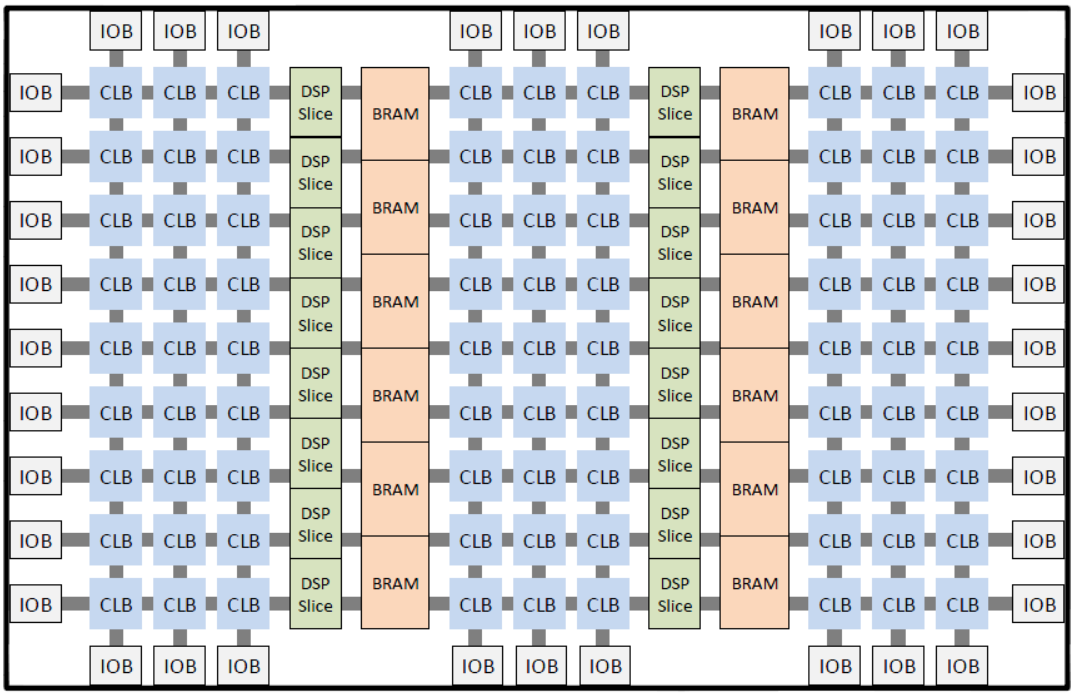


Figure 3.3: FPGA programmable logic [17]

MicroBlaze Soft-core Architecture

MicroBlaze is a 32-bit Harvard Reduced Instruction Set Computer (RISC) architecture optimized for synthesis and implementation into Xilinx FPGAs with a separate 32-bit instruction and data buses to execute programs and access data from both on-chip and external memory at the same time. Figure 3.4 presents a simple MicroBlaze soft-core [20, 18]. It has Harvard memory architecture and uses:

- two Local Memory Busses (LMB) for instruction and data memory
- two Block RAMs (BRAM)
- two peripherals connected via On-chip Peripheral Bus (OPB)

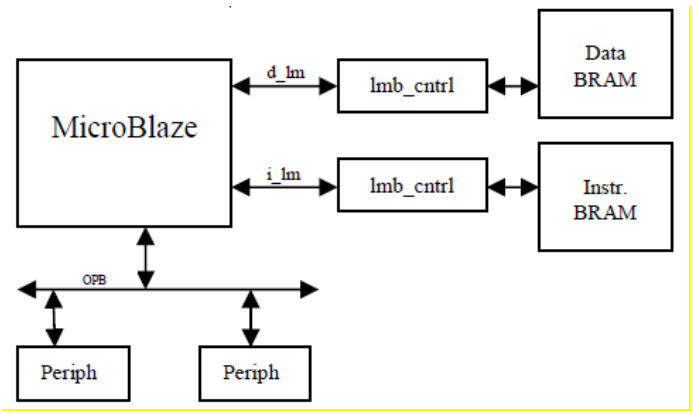


Figure 3.4: MicroBlaze Architecture [20]

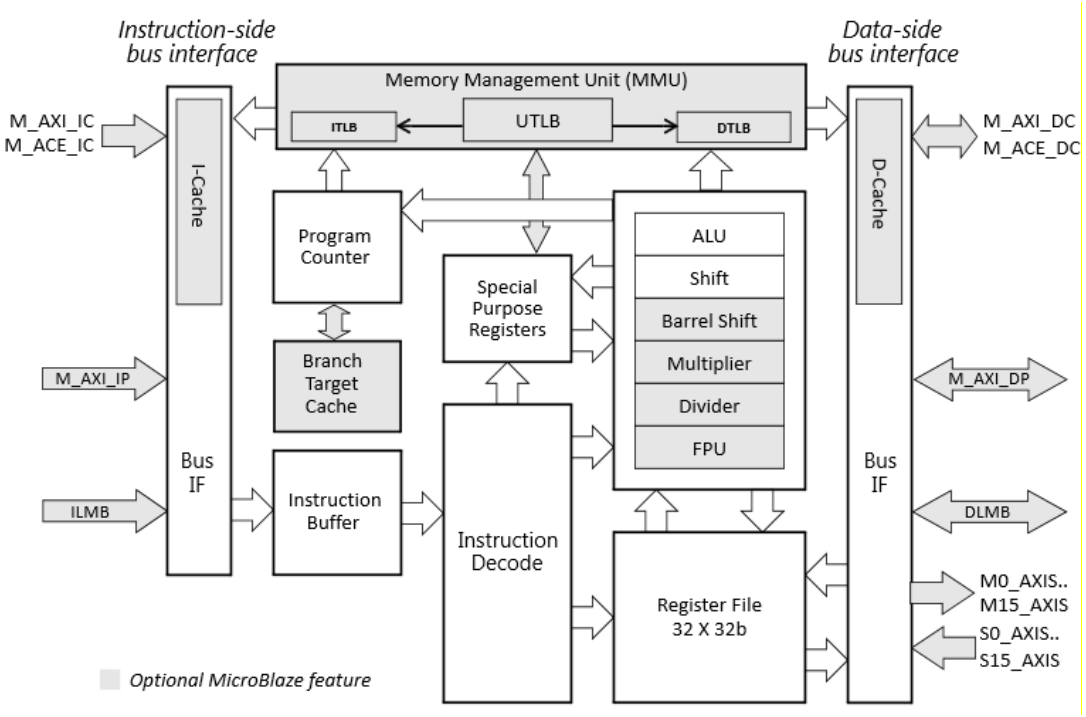


Figure 3.5: MicroBlaze Soft-Core Block Diagram [18]

The MicroBlaze soft-core offer to the designer flexibility during the design process and allows the designers to configure the microprocessor to meet the needs of their embedded systems with adding custom instructions, Intellectual Proprieties (IPs), particular coprocessor, etc. To increase the performance of the MicroBlaze, the designer can modify a number of features through the setting parameters. The configured parameters may include Integer Multiplier Units (mul), Barrel Shifter Units (BS), Integer Divider Units (ID), Floating Point Units (FPU), Machine Status Register Units (MSRU) and Pattern Compare Unit (PCU) [20].

We used Xilinx Vivado 2014.4 for configuring the FPGA and to include a MicroBlaze soft-core with an 8 KB cache memory (64KB is the maximum memory), 125 MHz (the maximum frequency possible). For each application tested, it was used the exact same MicroBlaze

configuration and the resulting system was analyzed in execution time metric. In the work of MHADHBI et al. [20] they used the metric of hardware area, which presents one of the metric in the choice of embedded systems which requires an optimal area. However, in real-time complex applications, both execution time, area and energy consumption determine the efficiency and the high performance of the configured embedded system.

The over 70 user-configurable options of the MicroBlaze enables any processor use case from a very small footprint state machine or microcontroller to a high performance compute-intensive microprocessor-based system running Linux. MicroBlaze can operate in either 3-stage pipeline mode to optimize size, or 5-stage pipeline mode to optimize speed delivering faster DMIPs performance than any other FPGA-based soft-processing solution (http://www.xilinx.com/tools/feature/csi/microblaze.htm). In Table 3.2 we can see the performance levels of the MicroBlaze Processor.

Device Family	Zynq-7000 SoC
Performance Optimized MicroBlaze with branch optimizations (5-stage pipeline) 1.38 DMIPs/MHz	228DMIPs
Performance Optimized MicroBlaze (5-stage pipeline) 1.30 DMIPs/MHz	259DMIPs
Area Optimized MicroBlaze (3-state pipeline) 1.03 DMIPS/MHz	196DMIPs

Table 3.2: MicroBlaze Processor v8.40.b Performance Levels (v14.4 XPS)

Benchmark Description

In this thesis workload selection was made mostly from the MiBench suite [7], whose source code is adapted to the ARM instruction set and is freely available to all researchers [8]. Additionally there were added some algorithms that are commonly used for benchmarking e.g. K-means, BlackScholes, image processing, etc.

The MiBench Suite consists of six categories:

- Automotive and Industrial Control
- Network
- Security
- Computer Devices
- Office Automation
- Telecommunications

The fact that these different categories provide various program characteristics enables researchers, in systems architecture, to examine the designs for a particular market segment.

Our main idea was to select one benchmark of each category and modify it to run on the accelerator. In some cases, where the code was simple it was easy but in some other cases the code was too complicated or too large to run on the accelerator and it was decided to search for different available source code in the same domain.

Below we will describe the benchmark from each category that was used.

Automotive – basicmath

Automotive and Industrial Control category [8] aims to show the use of embedded processors in embedded control systems. Typical applications are controllers in auto & moto industry, engine performance monitors and sensor systems.

The basic math test performs simple mathematical computations that often don't have hardware support in embedded processors. In this paper the cases of cubic function solving, integer square root and angle conversion from degrees to radians are used.

Network – Dijkstra

Network category [8] represents embedded processors in network devices like routers and switches. The work done by these embedded processors consists of shortest path calculations, tree, table lookups and data input/output.

The Dijkstra benchmark constructs a large graph in adjacency matrix representation and calculates the shortest path between every pair of nodes using recursion of *Dijkstra's* algorithm. The Dijkstra's algorithm is a well-known solution to the "shortest path" problem and completes in $O(n^2)$ time.

Security – SHA-1

The security category in MiBench suite [8] includes several common algorithms for hashing and data encryption/decryption.

The SHA-1 is the secure hash algorithm that produces a 160-bit (20-bytes) message digest for a given input. It is often used in the exchange of encryption/decryption keys and for generating digital signatures. Additionally it is used in the MD4 and MD5 hashing functions.

Consumer Devices

The category of consumer devices [8] focuses mainly on multimedia applications with the representative algorithms being the jpeg encoding/decoding, the image color format

conversion, the image dithering color palette reduction, the MP3 encode/decode and finally HTML typesetting.

Due to the complexity of the applications and the great amount of memory required any adaptation was not applicable so it was decided to use a simpler code. For this category representative source code for image processing was retrieved from the site of Burkardt [9]. Kernels about image denoise and image edge detection was used as workload.

Office Automation – StringSearch

The category of Office Automation [8] contains primarily text manipulation algorithms to represent machinery, like printers, fax machines and word processors.

The StringSearch benchmark searches for given words in phrases using the case insensitive comparison algorithm of Boyer Moore Horspool (https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore%E2%80%93Horspool_algorithm last accessed 15/10/2015).

Telecommunications – FFT/IFFT & CRC32

In this category benchmarks consist of encoding/decoding algorithms, frequency analysis and a checksum algorithm [8].

In our case we used the FFT/IFFT and CRC32 encoding kernels.

The FFT/IFFT benchmark performs a Fast Fourier Transform and the Inverse transform on an array of data. Fast Fourier transforms are used in digital signal processing to find contained frequencies in a given input signal. The input data that we used in our workload is ramp.

The CRC32 kernel performs a 32-bit Cyclic Redundancy Check (CRC) on a file. CRC checks are used to detect errors in data transmission.

In the workload selection we added some extra applications like Black & Scholes [9,10] and K-means clustering algorithm [11,12].

Black–Scholes–Merton

The Black–Scholes or Black–Scholes–Merton model is a mathematical model of a financial market containing certain derivative investment instruments. The model gives a theoretical estimate of the price of European-style stock market options (https://en.wikipedia.org/wiki/Black%E2%80%93Scholes_model last accessed 15/10/2015).

K-means clustering

The k-means clustering is a method of vector quantization that comes from the signal processing. It is popular for cluster analysis in data mining. The K-means clustering targets to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data into Voronoi cells. The problem is computationally difficult, considered NP-hard; however, there are efficient heuristic algorithms that are used and converge quickly to a local optimum (https://en.wikipedia.org/wiki/K-means_clustering last accessed 15/10/2015).

In the Table 3.3 we present the different kernels that were selected.

Cubic Solving
Unsigned Integer Square Root Computation
Degrees to Radians Conversion
BlackScholes Option price estimation
Dijsktra Shortest Path Algorithm
SHA-1
CRC32
FFT/IFFT
Image Denoise
Image Edge Detection
String Search
K-means Clustering

Table 3.3: List of Selected Kernels

Evaluation metrics

A hardware timer in the platform was used to measure execution cycles as a basis for evaluating execution time depended on the problem size. In all measurements we get the execution time in cycles. The duration of each cycle is 10 nsec which is 10^{-8} sec.

Another performance aspect is energy consumption. Sen and Wood [13] came to a conclusion that it is possible to increase system performance by decreasing cache size for a given power budget and they showed that workload reached performance improvements of 2-16%. In our case it was revealed, from the measurements, that execution time in cycles was higher when executing code without caching rather than with caching enabled. The model that could be followed is in some accelerators to disable caching in order to reduce energy consumption.

Experimental Results

In this section the results of the tests will be presented and explained. For each kernel there were different inputs and the execution time in cycles is shown in the graphs. There are two kinds of tests one with data cache enabled and one with data cache disabled.

Measurements Graphs

BasicMaths Cubic Solving

The Kernel Cubic test solves the third-degree polynomial equation, and serves to measure performance in performing typical mathematical operations (addition, multiplication, basic trigonometry).

The tests that were conducted for the computation of the third level equation $aX^3+bX^2+cX+d=0$ for different values of the a, b, c, d constants. The ranges of the constants are within the following limits -10<a<10, -10<b<10, -5<c<15, -3<d<3.

In this test we can see the results in execution time for cubic solving. As it can be seen in Figure 3.6 there are two levels of execution time. The lower level is when there is only one value as a result for the equation and the higher level is when there are three values that solve the third level polynomial equation.
In the case of an enabled caching the results are 100000 cycles when there is only one solution of the equation and 200000 cycles when there are three solution of the equation. In the other case when the caching was disabled the execution time in cycles for one solution is 400000 cycles and when there are solutions around 550000 cycles.
A sample of the results table is in the following Table 3.4.

6,5	a1=	-10	b1=	10	c1=	-5	d1=	3	Cycles	125193	Solutions:	1	x[0]=	0,832				
2	a1=	-10	b1=	10	c1=	-5	d1=	0,9	Cycles	125253	Solutions:	1	x[0]=	0,18				
3	a1=	-10	b1=	10	c1=	-5	d1=	-2,819	Cycles	122780	Solutions:	1	x[0]=	-0,31				
4	a1=	-10	b1=	10	c1=	-2,39	d1=	3	Cycles	125236	Solutions:	1	x[0]=	1,45				
5	a1=	-10	b1=	10	c1=	-2,39	d1=	0,9	Cycles	183570	Solutions:	3	x[0]=	0,46	x[1]=	0,657	x[2]=	0,296
6	a1=	-10	b1=	10	c1=	-2,39	d1=	-2,819	Cycles	124854	Solutions:	1	x[0]=	-0,374				
7	a1=	-10	b1=	10	c1=	0,22	d1=	3	Cycles	124649	Solutions:	1	x[0]=	1,219				
8	a1=	-10	b1=	10	c1=	0,22	d1=	1	Cycles	124222	Solutions:	1	x[0]=	1,29				
9	a1=	-10	b1=	10	c1=	0,22	d1=	-2,819	Cycles	124528	Solutions:	1	x[0]=	-0,448				
10	a1=	-10	b1=	10	c1=	2,829	d1=	3	Cycles	124495	Solutions:	1	x[0]=	1,367				

Table 3.4: Sample Results of Cubic Solving Tests

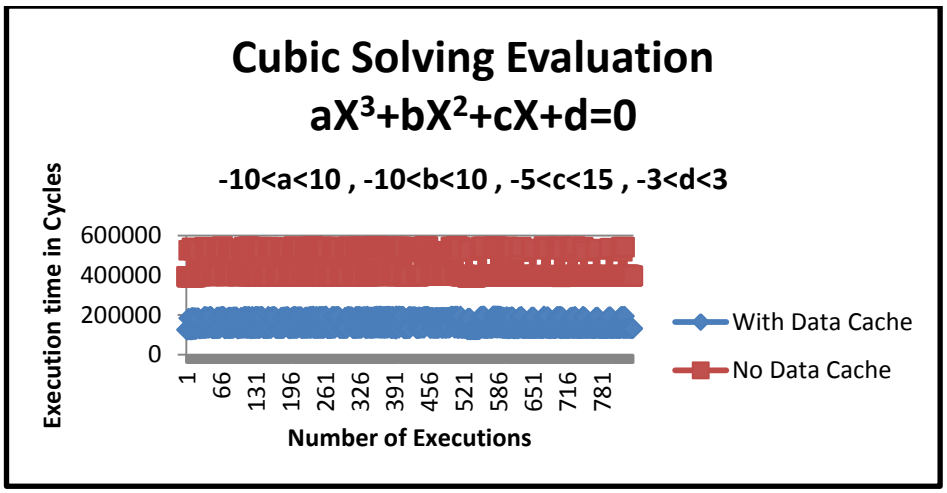


Figure 3.6: Execution times required for solving the equation $aX^3+bX^2+cX+d=0$ with constants a,b,c,d within the ranges -10<a<10 , -10<b<10 , -5<c<15 , -3<d<3

Unsigned Integer Square Roots Computation

The Kernel Unsigned Integer SQRT [8] finds the square root of an unsigned integer with all calculations being the base-two analogue of the square root algorithm. Bit-wise operations are the prevailing type in this test case with absolutely no multiplications or divisions.

As we can see in the Figure 3.7 when the caching is enabled, execution time is around 2000 cycles, for a wide range of integers (0 - 4FEBB399). On the other hand when caching is disabled the execution time is around 20000 cycles with the tendency to increment while the integer increases.

The Table 3.5 is a sample of the results of the Kernel USQRT(x).

Integer		Result	Desc	With Data Cache Cycles
0	sqrt(0) =	0	Total cycles	1777
5400	sqrt(5400) =	4815892	Total cycles	1867
10800	sqrt(10800) =	6810700	Total cycles	1894
16200	sqrt(16200) =	8341371	Total cycles	1930
21600	sqrt(21600) =	9631785	Total cycles	1876
27000	sqrt(27000) =	10768663	Total cycles	1867
32400	sqrt(32400) =	11796480	Total cycles	1813
37800	sqrt(37800) =	12741654	Total cycles	1867

Table 3.5 : Sample Result of Unsigned SQRT tests

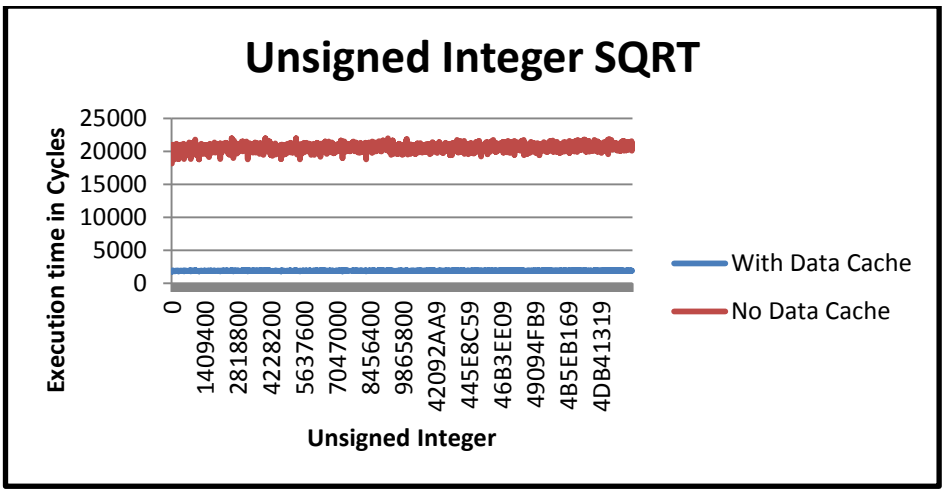


Figure 3.7: Execution times for computing the SQRT of Unsigned Integer(Unsigned Integer range within 0 - 4FEBB399)

Degrees to Radians Conversion

In this test the conversion of degrees to radians is applied. As we can see in Figure 3.8 the execution time for conversion is around 2700 cycles when caching is enabled and 7800 cycles when caching is disabled. The range of degrees converted was from 0^o to 500^o degrees. Normally the results of the conversion between 0^o – 360^o degrees would be enough but for testing reasons the tests conducted were within the range of 0^o – 500^o degrees in order to verify that there is no difference.

The Table 3.6 is a sample of the test performed for the Degrees to Radians Kernel.

0	degrees	0	radians	Total cycles	437
0,5	degrees	0,8	radians	Total cycles	2789
1	degrees	0,17	radians	Total cycles	2789
1,5	degrees	0,26	radians	Total cycles	2853
2	degrees	0,34	radians	Total cycles	2789
2,5	degrees	0,43	radians	Total cycles	2769
3	degrees	0,52	radians	Total cycles	2853
3,5	degrees	0,61	radians	Total cycles	2777
4	degrees	0,69	radians	Total cycles	2789
4,5	degrees	0,78	radians	Total cycles	2762

Table 3.6 : Sample of Deg2Rad(x) Tests

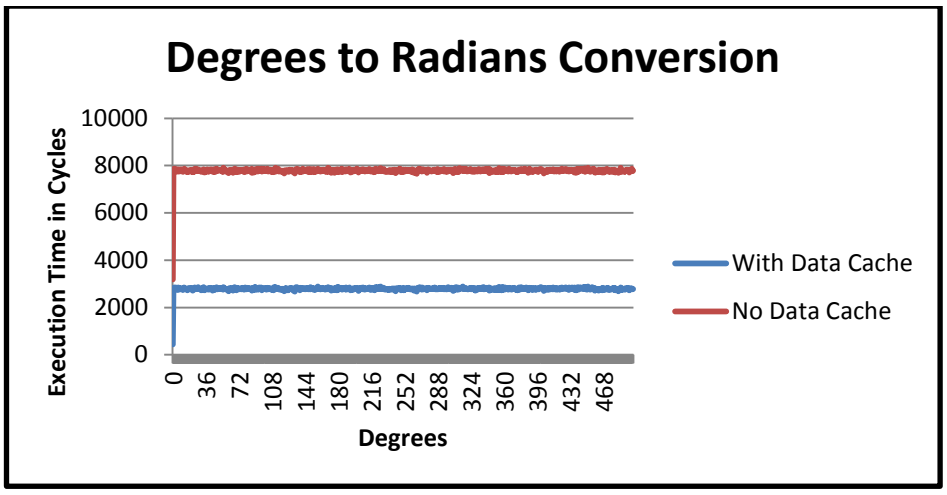


Figure 3.8: Execution Times for the conversion of Degrees to Radians Degrees Range within 0^{o}-500^{o}

Black & Scholes

In this application the price of an option is estimated for different number of days using the Black Scholes algorithm. It can be seen in Figure 3.9 that as the number of days increases the execution time accordingly increases too. It can also be observed that when caching is enabled the execution time is lower than when caching is disabled.

In the Table 3.7 there is a sample of the tests performed with Black Scholes Application.

	No of Days					Option Price
n=	10	Total cycles:	1078228	asset price	x[10]	2,209
n=	20	Total cycles:	2114111	asset price	x[20]	2,609
n=	30	Total cycles:	3277024	asset price	x[30]	1,327
n=	40	Total cycles:	4296202	asset price	x[40]	3,127
n=	50	Total cycles:	5480005	asset price	x[50]	1,436
n=	60	Total cycles:	6515138	asset price	x[60]	1,876
n=	70	Total cycles:	7553654	asset price	x[70]	2,400
n=	80	Total cycles:	8620348	asset price	x[80]	2,219
n=	90	Total cycles:	9787587	asset price	x[90]	2,237
n=	100	Total cycles:	11004253	asset price	x[100]	1,115

Table 3.7 : Sample results of the BlackScoles Tests

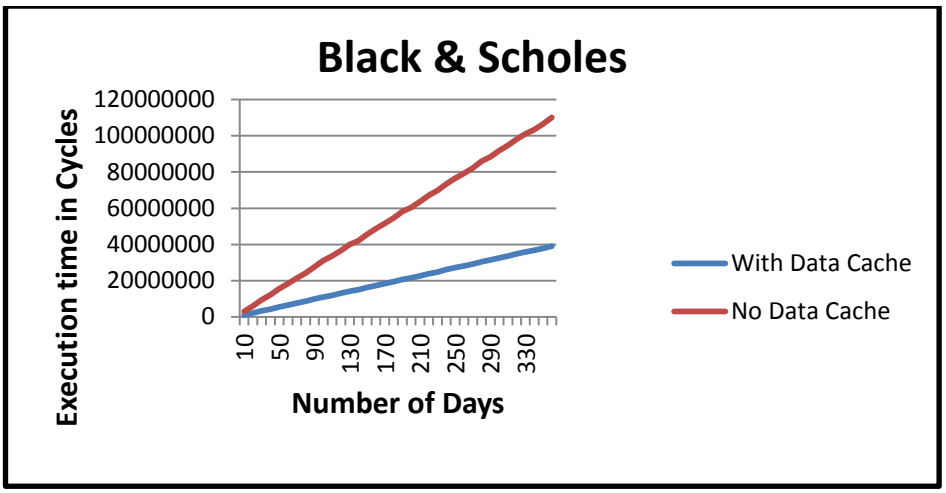


Figure 3.9: Execution Times for the computation of a stock price for a range of days within 10 – 365

CRC32

The 32-bit cyclic redundancy check (CRC) error-detecting code is performed in this kernel. As it can be seen the execution time depends on the size of the data that the check value is computed. As the data size increases, accordingly increases the execution time too. The Caching also affects the execution time. When caching is enabled the computation is faster. In Figure 3.10 the Data size ranges from 1 – 660 bytes.

In Table 3.8 there is a sample of the test performed for the CRC32 kernel.

	Data Size		Cycles	Check Value
Data Size	10	Total cycles:	791	DB63106E
Data Size	60	Total cycles:	3357	9F0875FE
Data Size	110	Total cycles:	5209	086BE43E
Data Size	160	Total cycles:	7521	D43CD9CC
Data Size	210	Total cycles:	9561	C304553C
Data Size	260	Total cycles:	11781	9DB4B794
Data Size	310	Total cycles:	13842	34D3D4AC
Data Size	360	Total cycles:	16037	10D92D68
Data Size	410	Total cycles:	18265	43D170C6
Data Size	460	Total cycles:	20206	202F21F8
Data Size	510	Total cycles:	22280	DBB45A49
Data Size	560	Total cycles:	24321	18B0D900
Data Size	610	Total cycles:	26354	772D238C
Data Size	660	Total cycles:	28257	EA7AB5B9

Table 3.8 : Sample Results of the CRC32 tests

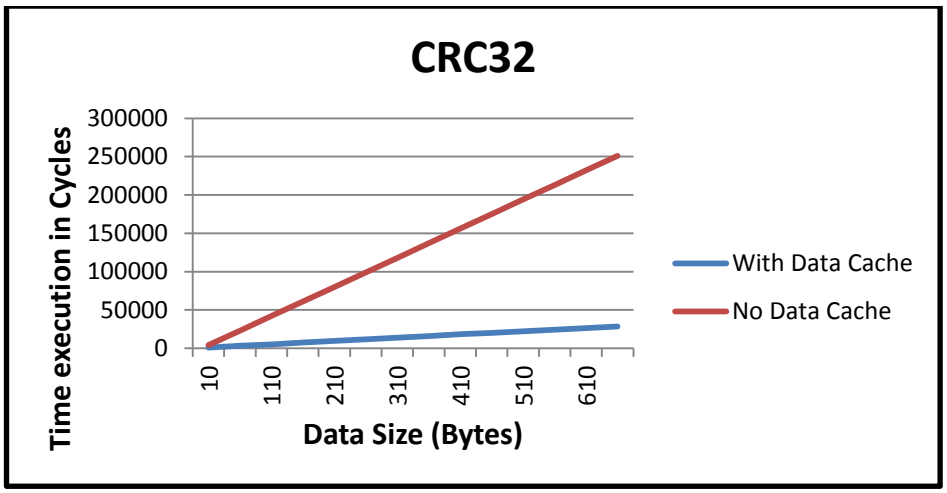


Figure 3.10: Execution Times for creating a CRC code for a range of text size within 10-660 bytes

Dijsktra

As it was mentioned in the benchmark description, the Dijkstra benchmark constructs a large graph in adjacency matrix representation and then calculates the shortest path between every pair of nodes using recursion of the Dijsktra's algorithm.

In Figure 3.11 the execution time in cycles is logged for different number of nodes in the graph. The scope was to log the time needed to find 5 shortest paths while the number of nodes increases. The result is that while the number of nodes increases the execution time in cycles is growing in exponential rate.

In Figure 3.12 execution time in cycles was logged for 50 nodes in the graph while the number of shortest paths increases by five (5). The number of shortest paths begins from 5 to 50. As we can observe in Figure 3.12 the time is the same for any number of shortest path, this can be explained from the fact that the algorithm in order to find the path needs to search the paths between all the nodes of the graph.

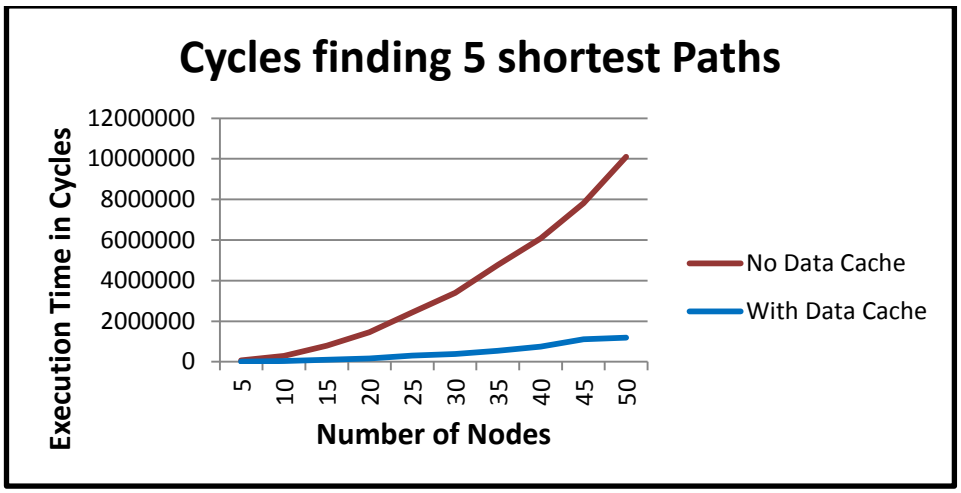


Figure 3. 11: Execution Times for finding 5 shortest paths in a Graph of adjacent Nodes within the range of 5 - 50

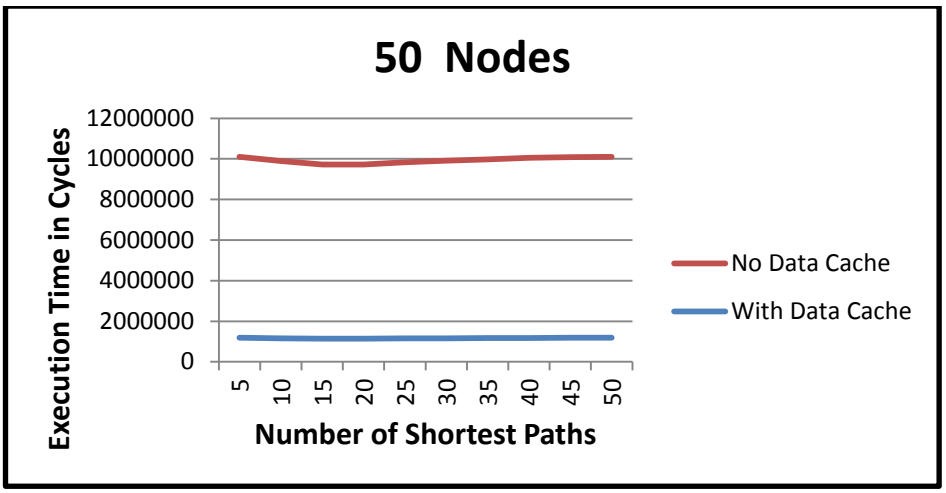


Figure 3.12: Execution Times to find 5 – 50 shortest paths in a Graph of 50 adjacent Nodes

SHA-1

In Figure 3.13 the execution time in cycles was logged for the computation of the message digest for different data size (25 - 9925 bytes) messages. The results show that the computation time increases proportionally while the length of the message is increasing too. The calculation rate is greater when cache is enabled in contrast when it is disabled.

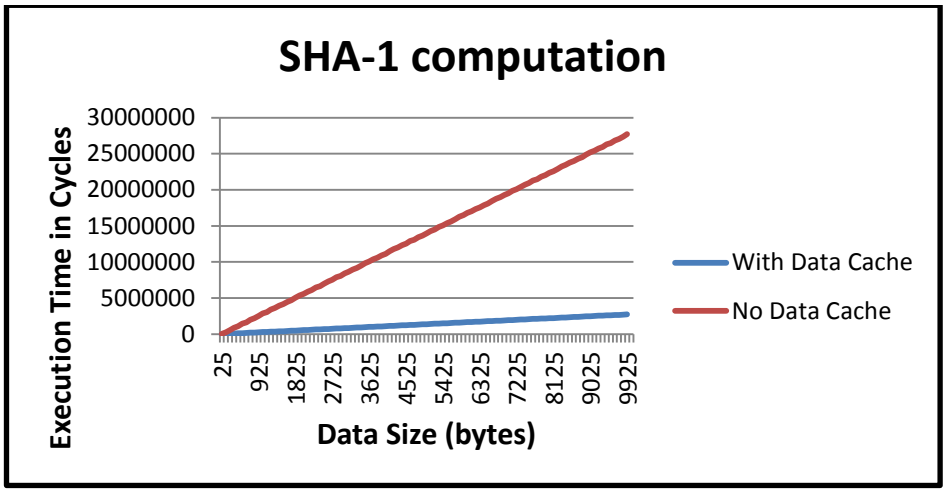


Figure 3.13: Execution Times to execute the hash function for a range of bytes within 25- 9925 bytes

In Table 3.9 it is a sample of the tests performed.

Bytes	225	Total cycles:	55806	BA9867F3	25C30980	CF079B71	FA07AAFC	40825A56
Bytes	325	Total cycles:	88918	D19A42B5	80FA6E6F	0B6200CF	3A3ADDEC	F5D170C7
Bytes	425	Total cycles:	109512	14BBADF2	3E4B62AD	A9E9D5ED	081C16B7	38B3EEB9
Bytes	525	Total cycles:	142624	939DC1C8	C800E63F	53590F5C	6275E765	9DD3CB73
Bytes	625	Total cycles:	163218	D5DF091E	56E304DF	1555D728	6B694FE7	3FD93F8C
Bytes	725	Total cycles:	196330	E7250304	934CFACE	56CBE4E2	62E45BF1	CBECAE04
Bytes	825	Total cycles:	216924	4220EB0C	F3D29879	4AC5A934	685BC243	647BF7D2
Bytes	925	Total cycles:	250036	7DAAE909	1D1970A9	3478C790	76A78506	BEDA10F8

Table 3.9: Sample Results of SHA-1 tests

Digital Signal Processing
FFT & IFFT

Fast Fourier transforms are widely used for many applications in engineering, science, and mathematics. Gilbert Strang (1994) described the fast Fourier transform as *"the most important numerical algorithm of our lifetime"* and it was included in Top 10 Algorithms of 20th Century by the IEEE journal Computing in Science & Engineering (https://en.wikipedia.org/wiki/Fast_Fourier_transform last accessed 15/10/2015). If we perform a search on the Google for the words "FFT algorithms" the results will yield almost 330.000 web pages.

The Fast Fourier Transform (FFT) is an algorithm that computes the Discrete Fourier Transform (DFT) of a sequence, or it's inverse. The Fast Fourier Transform (FFT) is a faster version of the Discrete Fourier Transform (DFT). The FFT utilizes some sophisticated algorithms to do the same thing as the DFT, but very fast. The Fourier analysis is the conversion of a signal from its original domain time or space, to the frequency domain and the reverse conversion. This results to the reduction of computing complexity of the DFT from $O(\mathbf{n}^2)$ to $O(\mathbf{n} \log \mathbf{n})$ where **n** is the data size.

The kernel code written in C language was retrieved from https://github.com/prst/TM4C1294/blob/master/FFT.c (last accessed 15/10/2015) which was tweaked to run on the MicroBlaze accelerator as a function. The tests performed were ramps with an increasing number of samples. From the samples, the algorithm finds their FFT values and the original sequence using the inverse FFT. In Figure 3.14 we can see the results of the tests.

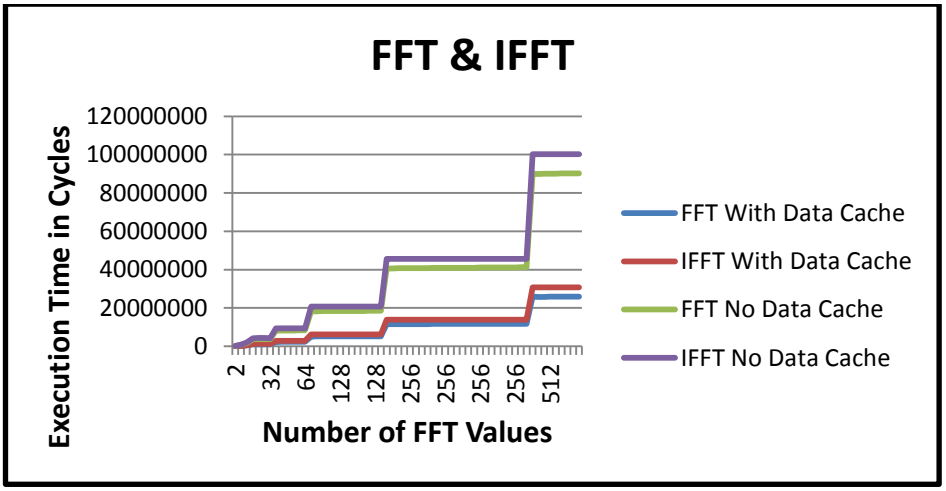


Figure 3.14: Execution Times for computing FFT & IFFT values for a range within 2 – 512

In Table 3.10 it can be seen the execution times for computing the FFT values of an increasing number of samples of the ramps that were created and the inverse process the creation of the original sequence.

Nx =	2	NFFT =	2	Total cycles for FFT:	31186	Total cycles for IFFT:	33754
Nx =	7	NFFT =	8	Total cycles for FFT:	204701	Total cycles for IFFT:	246652
Nx =	12	NFFT =	16	Total cycles for FFT:	457306	Total cycles for IFFT:	564867
Nx =	17	NFFT =	32	Total cycles for FFT:	1008303	Total cycles for IFFT:	1266023
Nx =	22	NFFT =	32	Total cycles for FFT:	1014936	Total cycles for IFFT:	1278634
Nx =	27	NFFT =	32	Total cycles for FFT:	1019225	Total cycles for IFFT:	1271897
Nx =	32	NFFT =	32	Total cycles for FFT:	1024358	Total cycles for IFFT:	1261734
Nx =	37	NFFT =	64	Total cycles for FFT:	2265023	Total cycles for IFFT:	2815385
Nx =	42	NFFT =	64	Total cycles for FFT:	2279737	Total cycles for IFFT:	2815796

Table 3.10 : Samples of the FFT tests

Image Processing

Image Denoise

The Image Denoise is a kernel written in C Language by Burkart [9]. According to the developer, the kernel uses the median filter to try to remove noise from an image. The gray scale image is represented by using a two dimensions (2D) array of positive integers over some range 0 to GMAX. The value 0 indicates black pixels, and GMAX white pixels. Intermediate values represent pixels with shades of gray in a natural way. Accordingly, a color image can be represented by using a set of three two dimensions (2D) arrays, which R, G, and B stand for the intensity of the red, green and blue signals which form the color image. The common maximum value could be assumed as the RGBMAX.

In our test cases ASCII PGM format were used due to the fact that they are convenient to pass as inputs parameters in the function that was created. The noise ("salt and pepper") was applied on images of different sizes. That is because a scattering of individual pixels have been reset to the lowest or highest possible values in a random way. In a gray scale picture, such noise looks like salt and pepper that was added to the picture.

Burkart [9] explains how the algorithm works, since an image is in a large degree smooth, each pixel should actually be close enough to the values of pixels nearby, which is not true for the salt and pepper pixels. So a way to make the noise go away is to replace each pixel by the median value of itself and its neighbors. During the tests conducted, it was noticed that when the filter was implemented more than once, the image was looking more and more to the original image. The method used in this kernel is very simple in contrast to other more

sophisticated filters. The code used is available here http://people.sc.fsu.edu/~jburkardt/c_src/image_denoise/image_denoise.html (last accessed 15/10/2015)

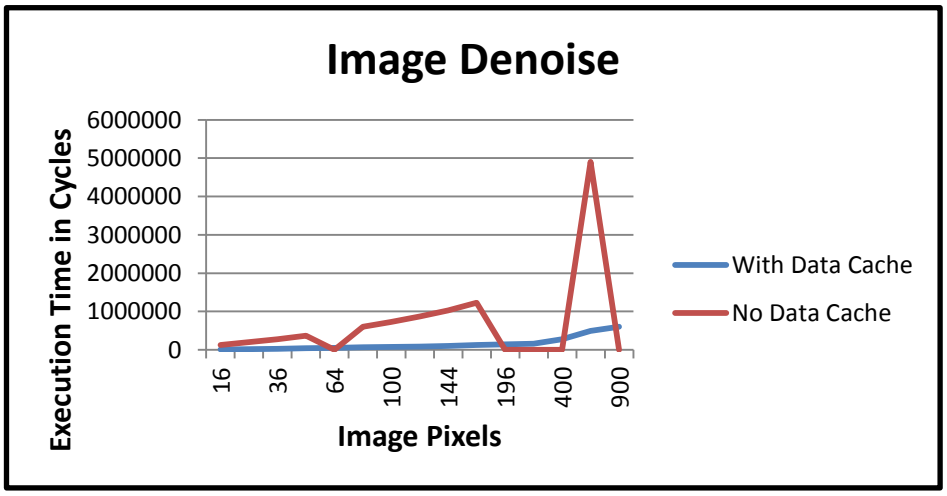


Figure 3.15 : Execution Times for applying median filter on images with image size with 16 - 900 pixels

In Table 3.16 can be seen a sample of the test applying median filter on differnet image size pictures.

Number of rows =	4	Number of columns =	4	Maximum pixel intensity =	255	Cycles	12774
Number of rows =	5	Number of columns =	5	Maximum pixel intensity =	255	Cycles	20091
Number of rows =	6	Number of columns =	6	Maximum pixel intensity =	255	Cycles	28087
Number of rows =	7	Number of columns =	7	Maximum pixel intensity =	255	Cycles	45370
Number of rows =	8	Number of columns =	8	Maximum pixel intensity =	255	Cycles	49164
Number of rows =	9	Number of columns =	9	Maximum pixel intensity =	255	Cycles	66311
Number of rows =	10	Number of columns =	10	Maximum pixel intensity =	255	Cycles	73279
Number of rows =	11	Number of columns =	11	Maximum pixel intensity =	255	Cycles	87746
Number of rows =	12	Number of columns =	12	Maximum pixel intensity =	255	Cycles	102624
Number of rows =	13	Number of columns =	13	Maximum pixel intensity =	255	Cycles	122663
Number of rows =	14	Number of columns =	14	Maximum pixel intensity =	255	Cycles	139874
Number of rows =	15	Number of columns =	15	Maximum pixel intensity =	255	Cycles	156543
Number of rows =	20	Number of columns =	20	Maximum pixel intensity =	255	Cycles	276893
Number of rows =	25	Number of columns =	25	Maximum pixel intensity =	255	Cycles	489624
Number of rows =	30	Number of columns =	30	Maximum pixel intensity =	255	Cycles	602092

Figure 3.16: Sample of Image Denoise Kernel tests

Image Edge Detection

Burkart [9] is also the developer of the Image edge detection kernel. The kernel is written in C language and detects the edges in an image. The algorithm applies the N.E.W.S. which is a very simple edge detection scheme, which compares the North, East, West, and South neighbors of a pixel in order to determine if the pixel lies along an edge. In the literature we

can find many sophisticated edge detection algorithms, but this was very suitable for the tests we needed to do. The C code was modified in order to be executed on the MicroBlaze microprocessor as a function. The execution times of the kernel were logged for different image size pictures. The results are depicted in Figure 3.17. The code of the kernel is available here http://people.sc.fsu.edu/~jburkardt/c_src/image_edge/image_edge.html (last accessed 14/10/2015).

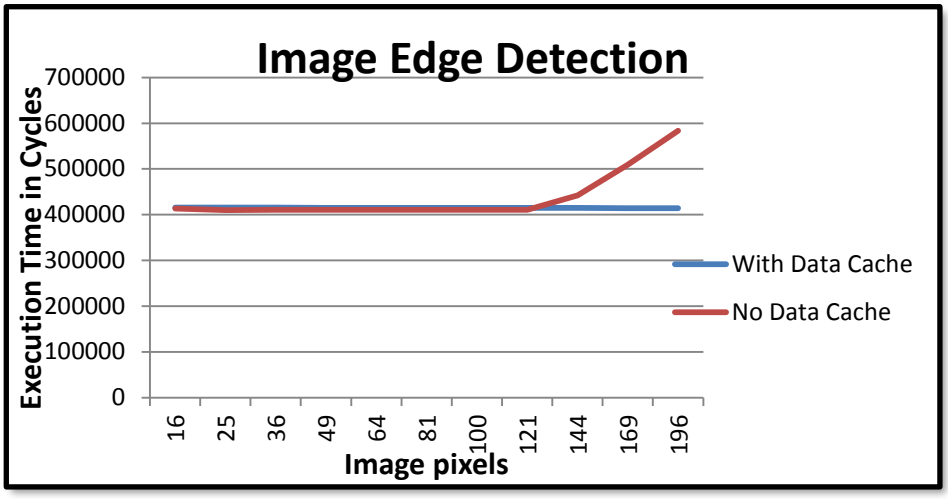


Figure 3.17: Execution Times for applying Edge Detection on Images with image size within 16 – 196 pixels

The Table 3.11 is a sample of the tests performed.

Rows=	4	Cols=	4	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	415639
Rows=	5	Cols=	5	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	415336
Rows=	6	Cols=	6	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	415303
Rows=	7	Cols=	7	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	415169
Rows=	8	Cols=	8	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	415089
Rows=	9	Cols=	9	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	414938
Rows=	10	Cols=	10	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	414927
Rows=	11	Cols=	11	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	414762
Rows=	12	Cols=	12	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	414604
Rows=	13	Cols=	13	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	414463
Rows=	14	Cols=	14	Maximum pixel intensity =	255	E_MAX =	245	THRESH =	49	Cycles=	414320

Table 3.11 : Sample of Image Edge Detection

Search String

The Boyer–Moore–Horspool algorithm or Horspool's algorithm is an algorithm for finding substrings in strings and was published by Nigel Horspool in 1980 [21]. It is a simplification of the Boyer–Moore string search algorithm which is also related to the Knuth–Morris–Pratt algorithm. The algorithm trades space for time in order to achieve a complexity of O(N) on

random text rather than O(MN) in the worst case, where the length of the searching words is M and the length of the search string is N (https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore%E2%80%93Horspool_algorithm).

In our test cases we performed multiple tests of the string search algorithm. The searches were different substrings sizes in a string whose size was increasing. This led to the results of execution times in cycles that depended on two independent variables, the string size and the substring size. The Figures 3.12 & 3.19 depict the 3D results of execution times depended on the substring and the string size. In Figure 3.12 can be seen the results for the case of caching enabled and in the Figure 3.19 are the results for caching disabled.

The red points in Figure 3.18 are values that are excluded from the model for predicting execution times. Two regions of points are excluded. The first Region contains the execution times that are produced from the algorithm and they reach the lower bounds of the model and the second region contains the execution times where the string to be found is larger than the string to be searched. These two regions are excluded due to the fact that we need the higher bound of the model in order to predict the Worst Case Execution Time. The regions can be seen in Figures 3.18 and 3.21.

The Figures 3.18 and 3.19 depict diferrent aspects of the 3D Figure 3.12.

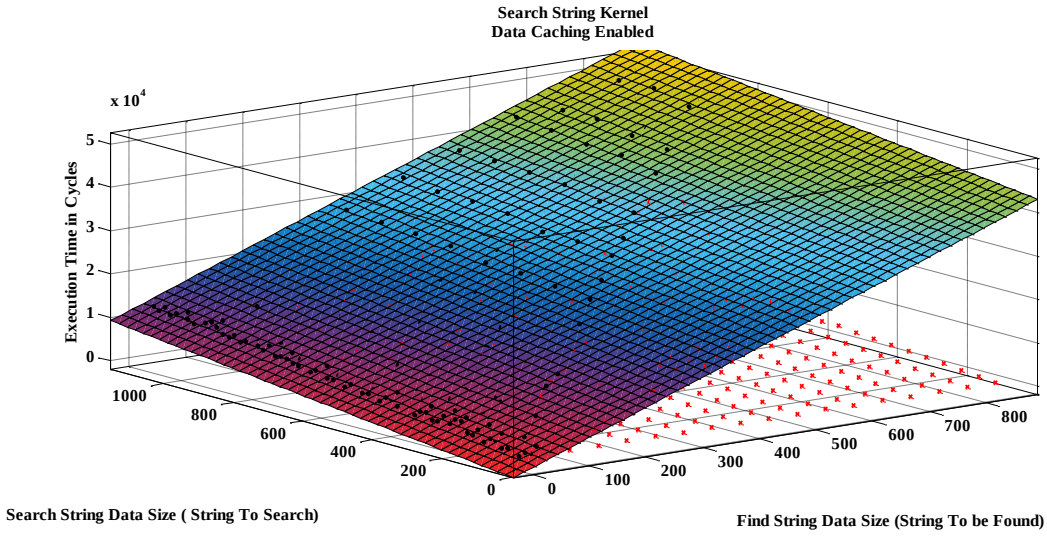


Table 3.12:Execution Times for searching strings within 50-1000 chars in strings within 50-1000 chars(Caching Enabled)

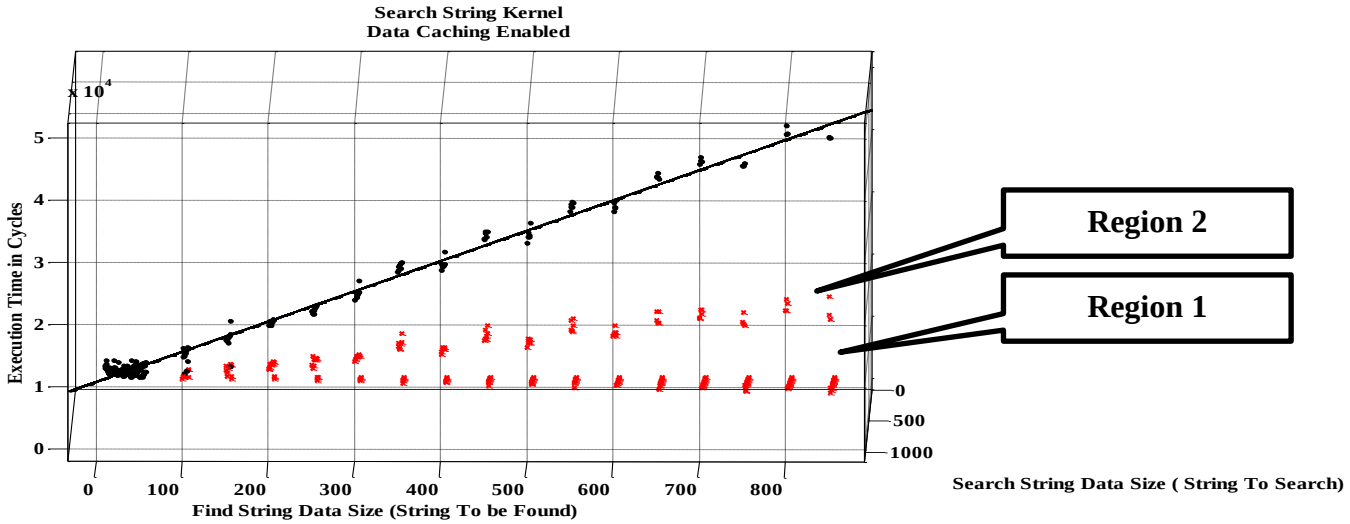


Figure 3.18: Different aspect of the 3D Figure 3.12 with Region 1 and 2

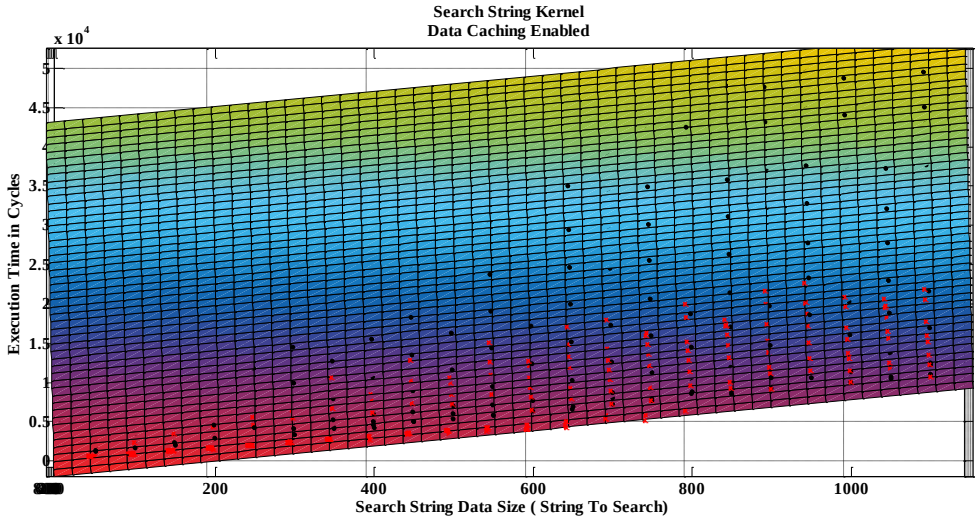


Figure 3.19: Different aspect of the 3D Figure 3.12

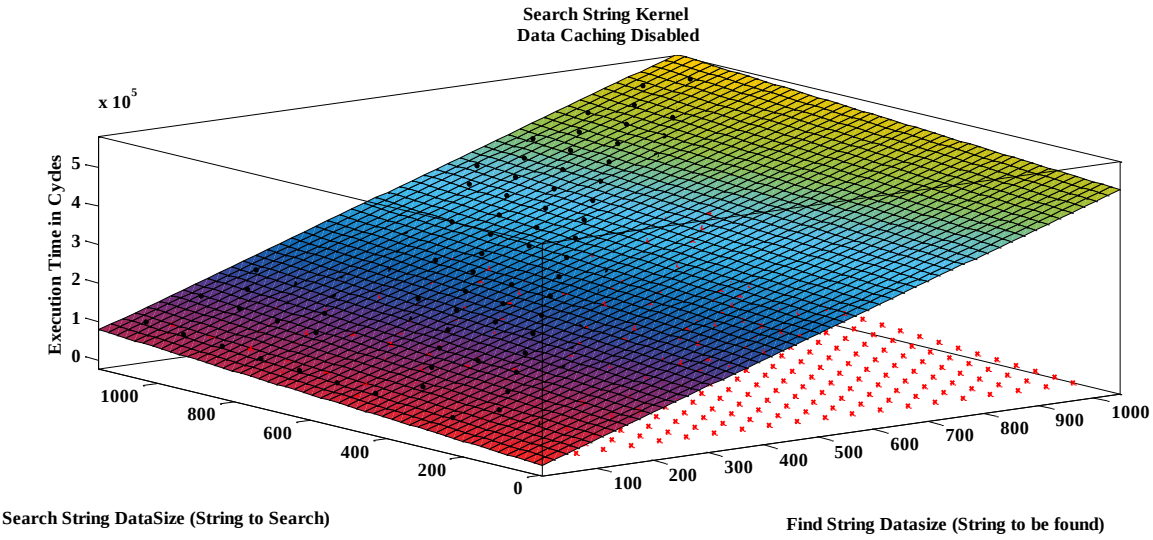


Figure 3.20: Execution Times for searching strings within 50-1000 chars in strings within 50-1000 chars (Caching Disabled)

The Figures 3.21 and 3.22 depict different aspects of the 3 Dimensional Figure 3.20.

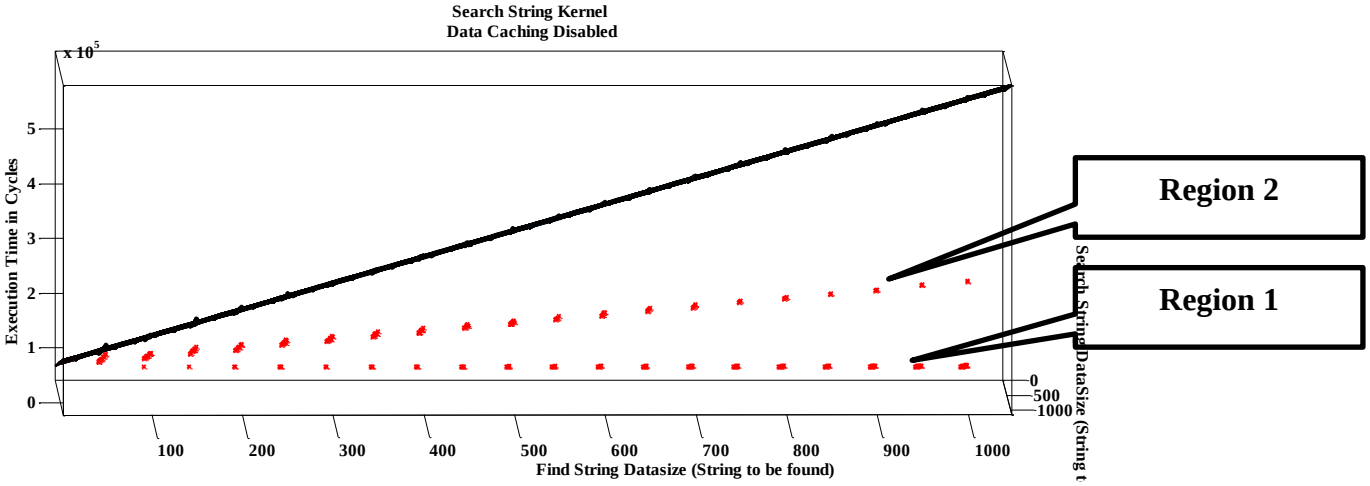


Figure 3. 21: Different aspect of the 3D Figure 3.15 with Region 1 and 2

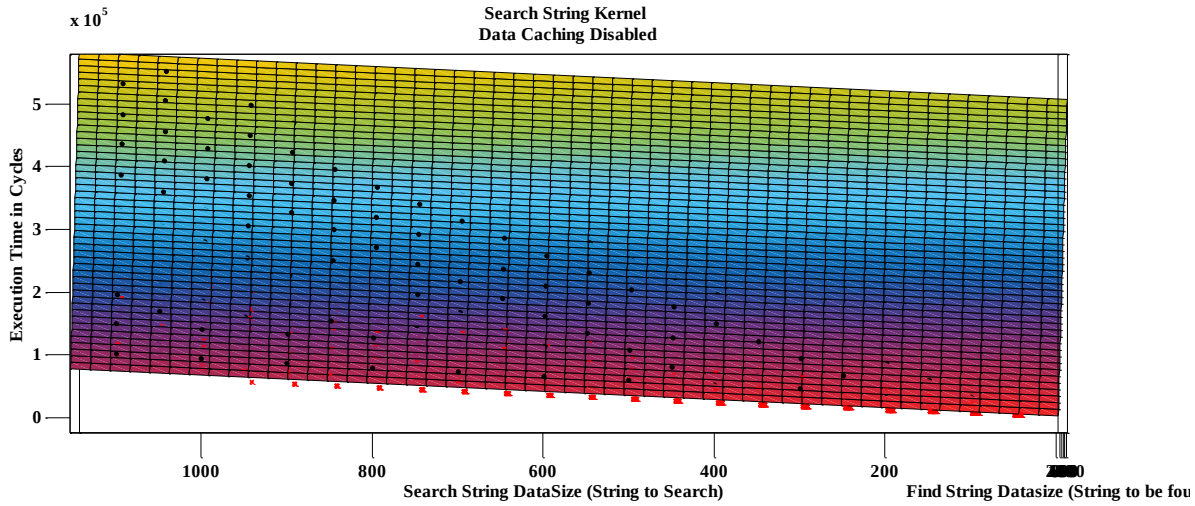


Figure 3.22: Different aspect of the 3D Figure 3.15

K-means Clustering

The K-means clustering is a method of vector quantization which originates from signal processing and is very popular for cluster analysis in data mining. The K-means clustering algorithm is defined by Hartigan [55]. The aim of the K-means algorithm is to divide M points with N dimensions into K clusters so that the within-cluster sum of squares is minimized. It is not practical to require that the solution has minimal sum of squares against all partitions, except when M, N are small and K equals to 2. The aim of the algorithm is to find those solutions where no movement of a point from one cluster to another will reduce the within-cluster sum of squares [54]. The problem is computationally difficult, NP-hard, but there are efficient heuristic algorithms that are commonly applied and meet quickly to a local optimum.

For the tests that were performed in this kernel modeling was the most difficult issue due to the fact that there were three independent variables to search for the relationships among them. The variables were the number of coordinates, the number of clusters and the number of objects. In order to conduct the tests, the number of coordinates was held fixed and we logged the execution times for a range of objects (60-120 objects) and a range of clusters (10-60 clusters). The procedure was repeated for a range of fixed coordinates (2-5 coordinates). The test results are in the Figures 3.23 – 3.46 for the case of data cache enabled and for data cache disabled.

Objects with Number of Coordinates 2 and Data Cache Enabled

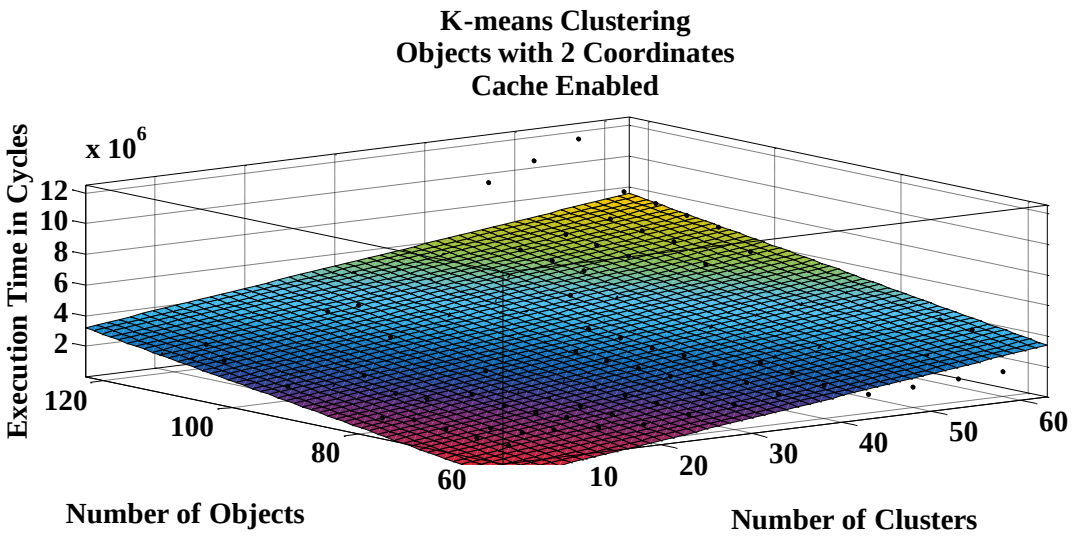


Figure 3.23 : Execution Times for K-means Clustering for Objects with Coordinates 2 Number of Objects within 60-120 Number of Clusters 10 – 60 (Caching Enabled)

Figures 3.24 and 3.25 are different aspects of 3 Dimensional Figure 3.23.

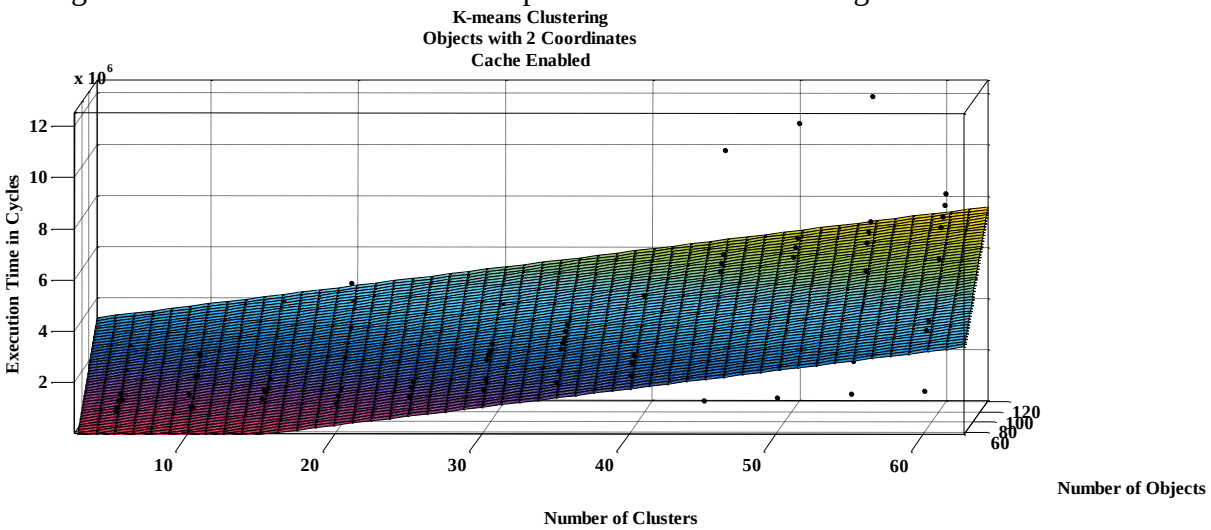


Figure 3.24 : Different aspect of 3D Figure 3.23

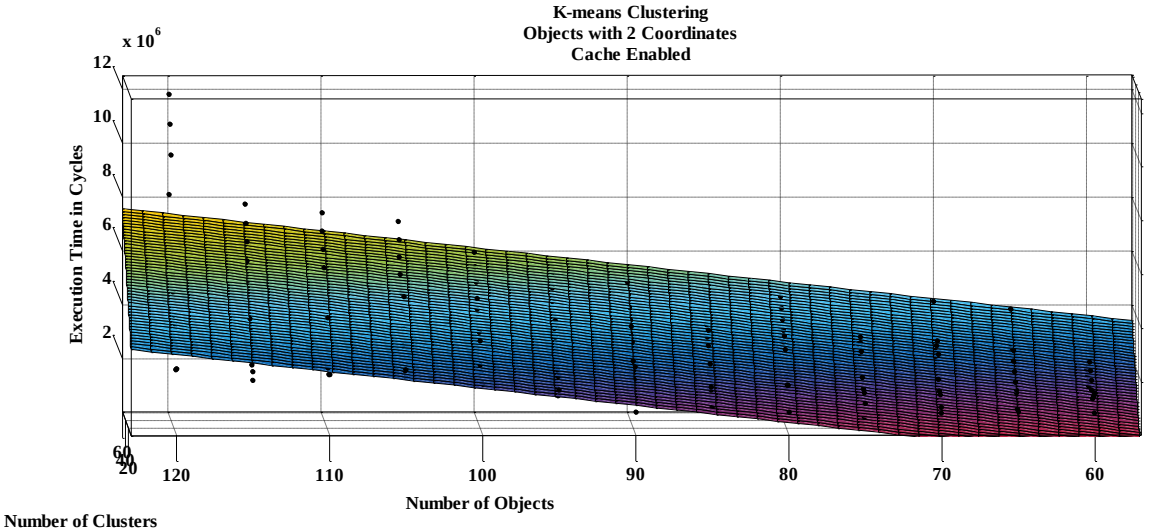


Figure 3. 25 : Different aspect of 3D Figure 23

From the Figures we can conclude that as the Number of Objects and the Number of Clusters increase then the execution time increases.

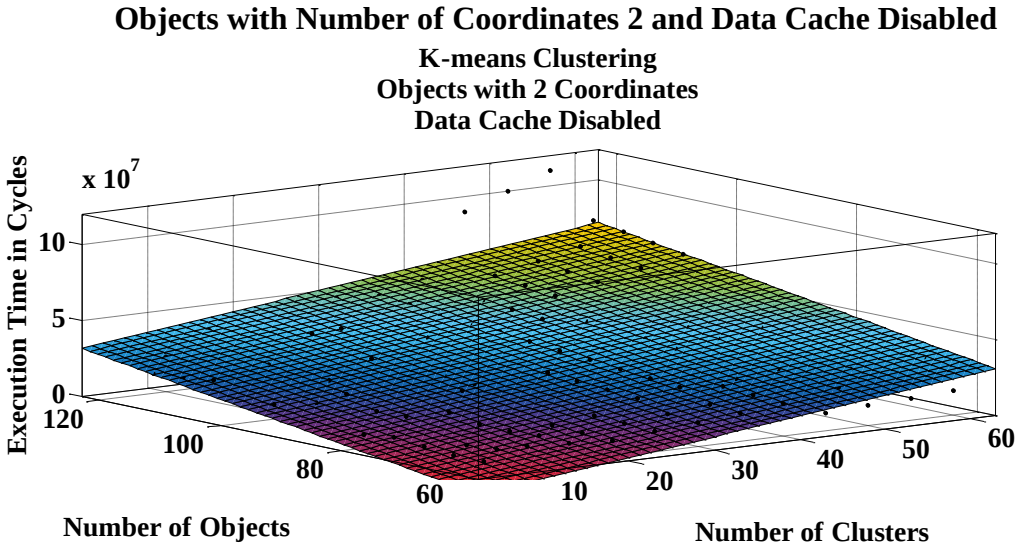


Figure 3. 26 : Execution Times for K-means Clustering, for Objects with Coordinates 2, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Disabled)

Figures 3.27 and 3.28 are different aspects of 3 Dimensional Figure 3.26.

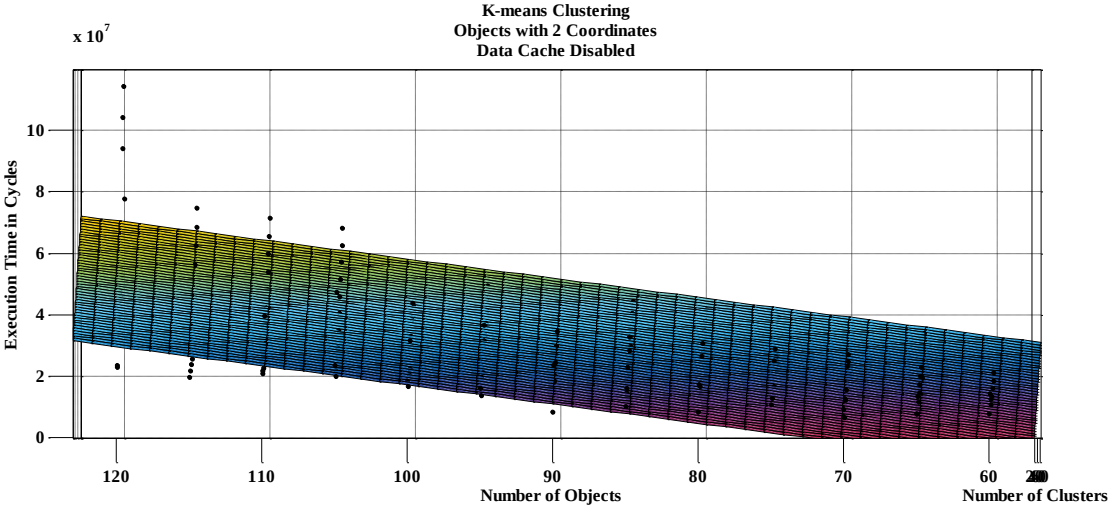


Figure 3. 27 : Different aspect of 3D Figure 3.26

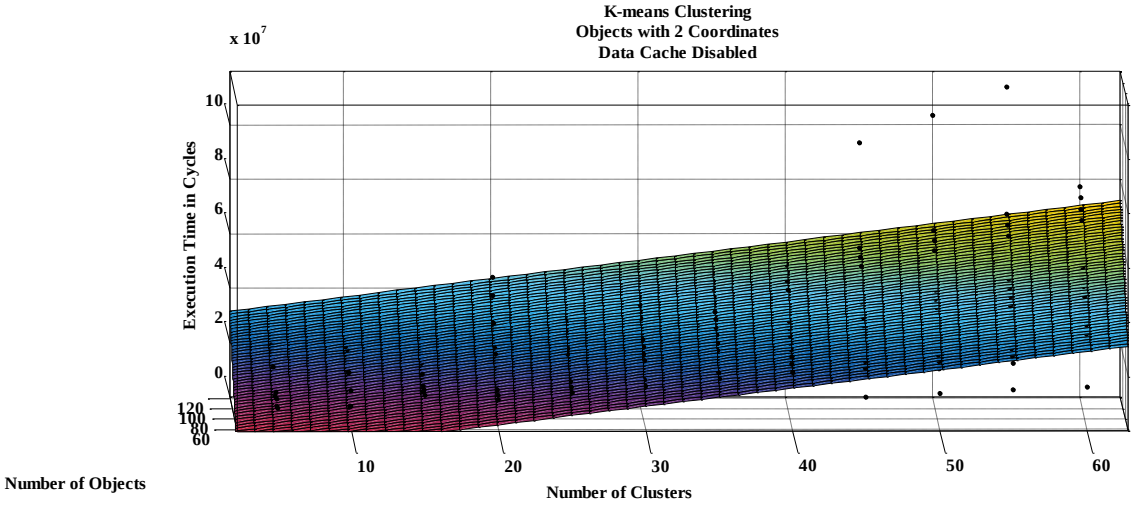


Figure 3. 28 : Different aspect of 3D Figure 3.26

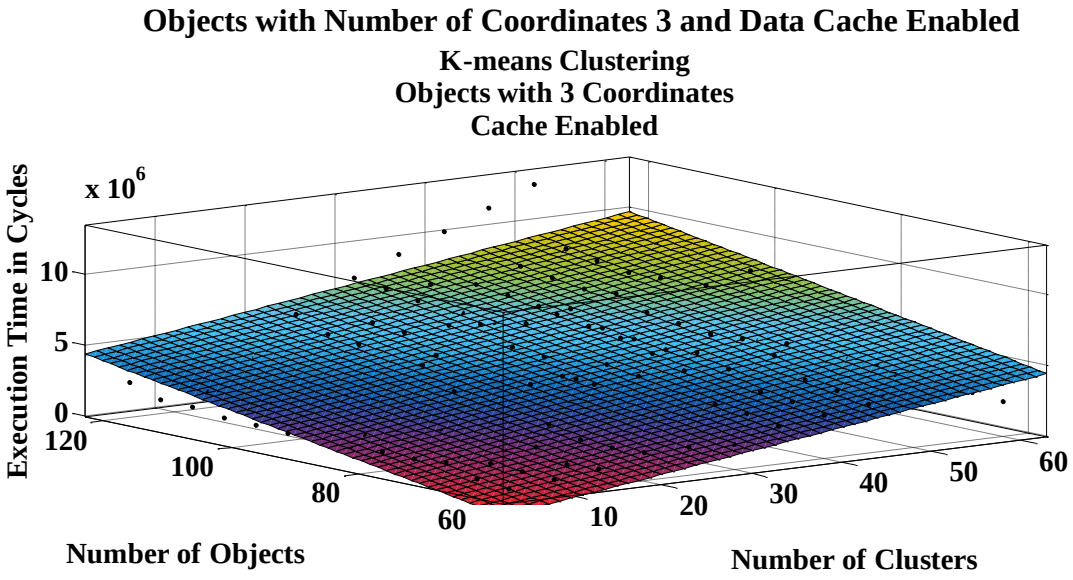


Figure 3. 29 : Execution Times for K-means Clustering for Objects with Coordinates 3 Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Enabled)

Figures 3.30 and 3.31 are different aspects of 3 Dimensional Figure 3.29.

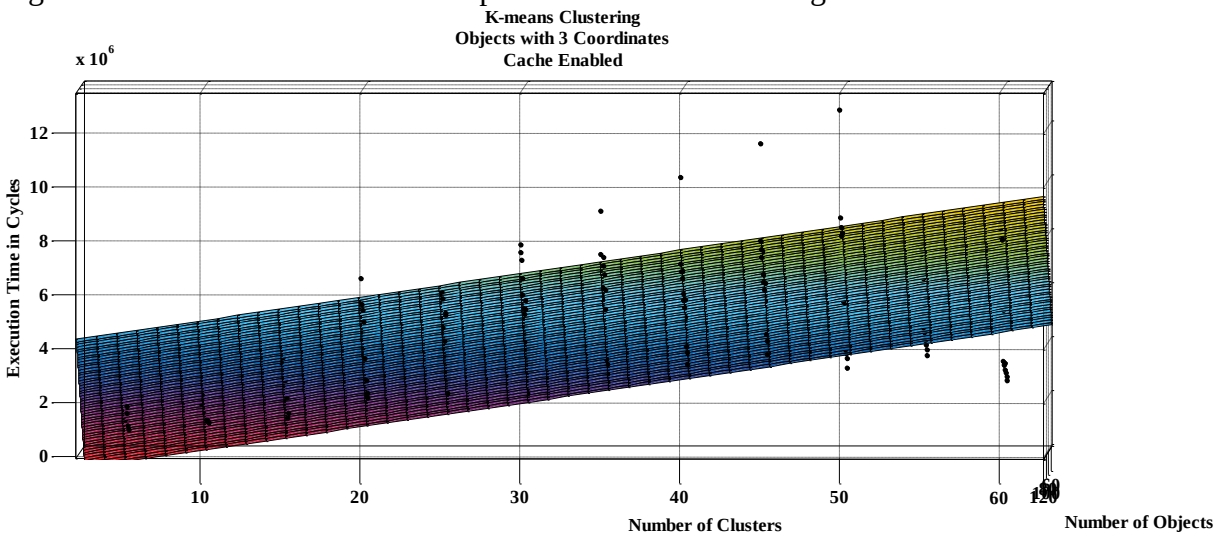


Figure 3. 30 : Different aspect of 3D Figure 3.29

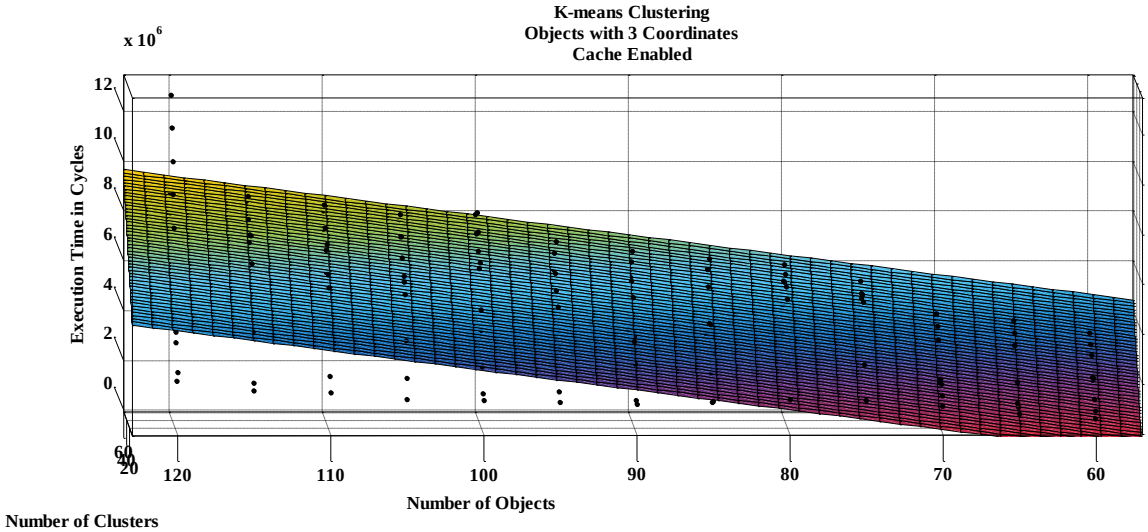


Figure 3. 31 : Different aspect of 3D Figure 3.29

Objects with Number of Coordinates 3 and Data Cache Disabled

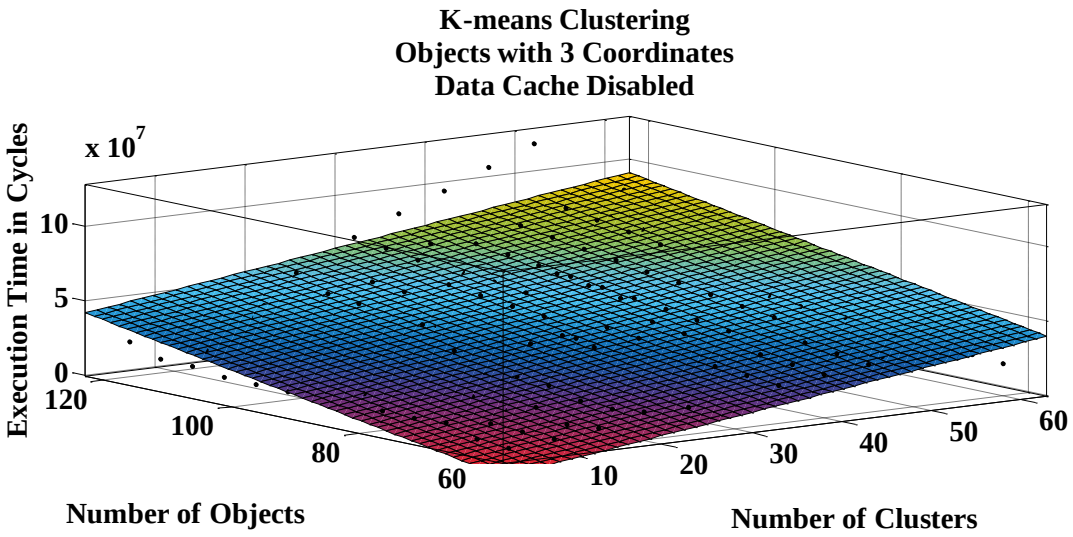


Figure 3. 32 : Execution Times for K-means Clustering for, Objects with Coordinates 3, Number of Objects within 60-120,Number of Clusters 10 – 60 (Caching Disabled)

Figures 3.33 and 3.34 are different aspects of 3 Dimensional Figure 3.32

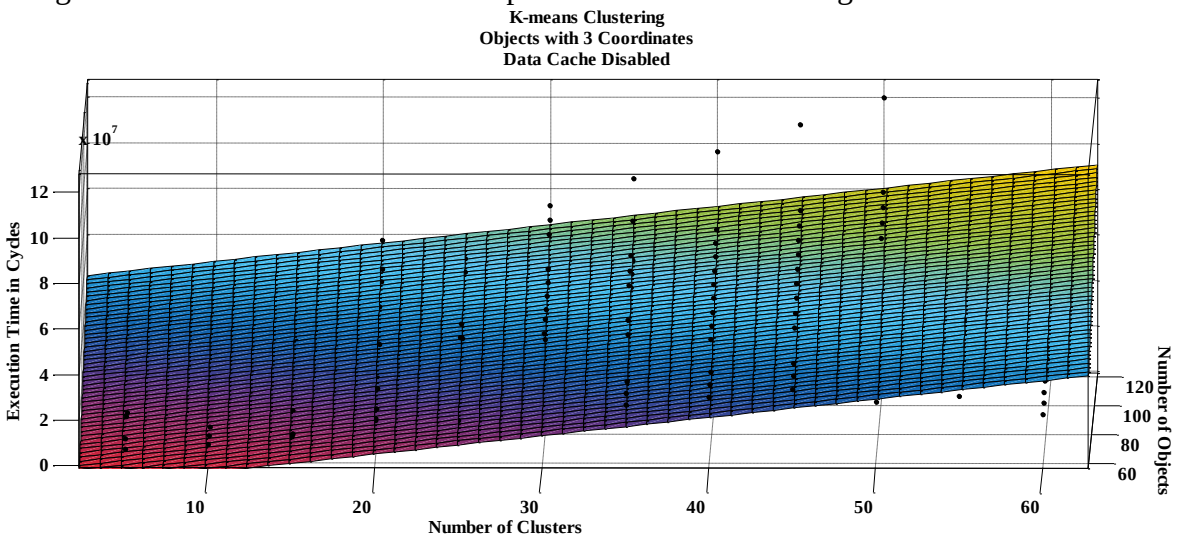


Figure 3. 33 : Different aspect of 3D Figure 3.32

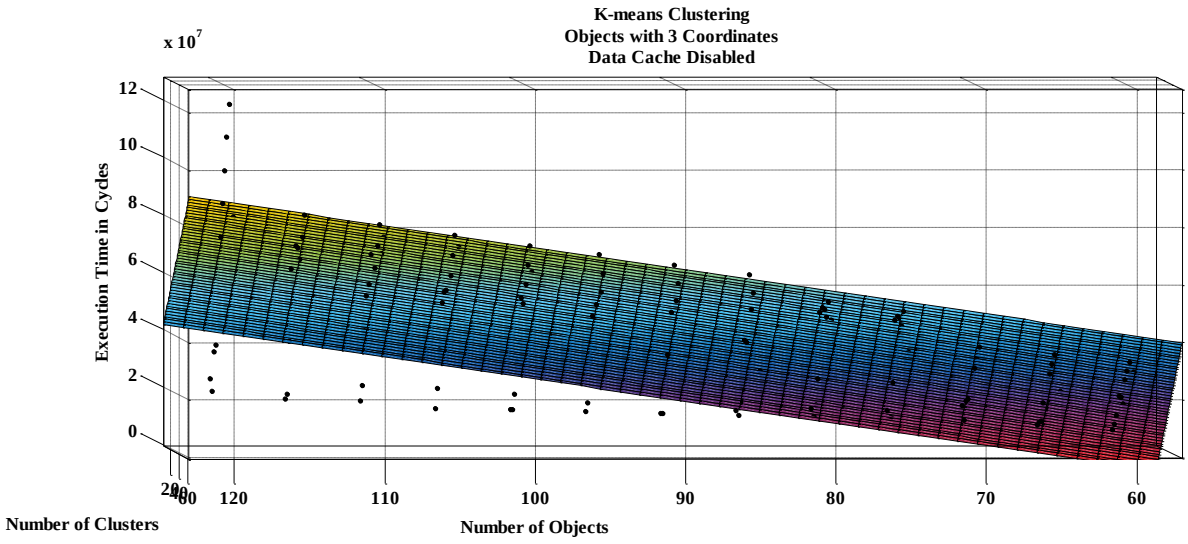


Figure 3. 34 : Different aspect of 3D Figure 3.32

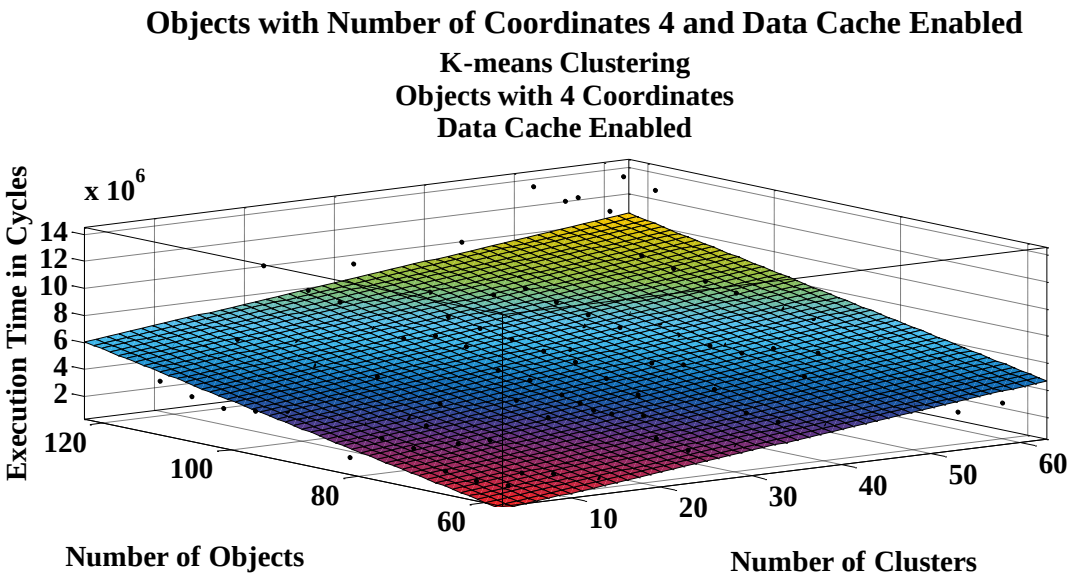


Figure 3. 35 : Execution Times for K-means Clustering for Objects with Coordinates 4, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Enabled)

Figures 3.36 and 3.37 are different aspects of 3 Dimensional Figure 3.35

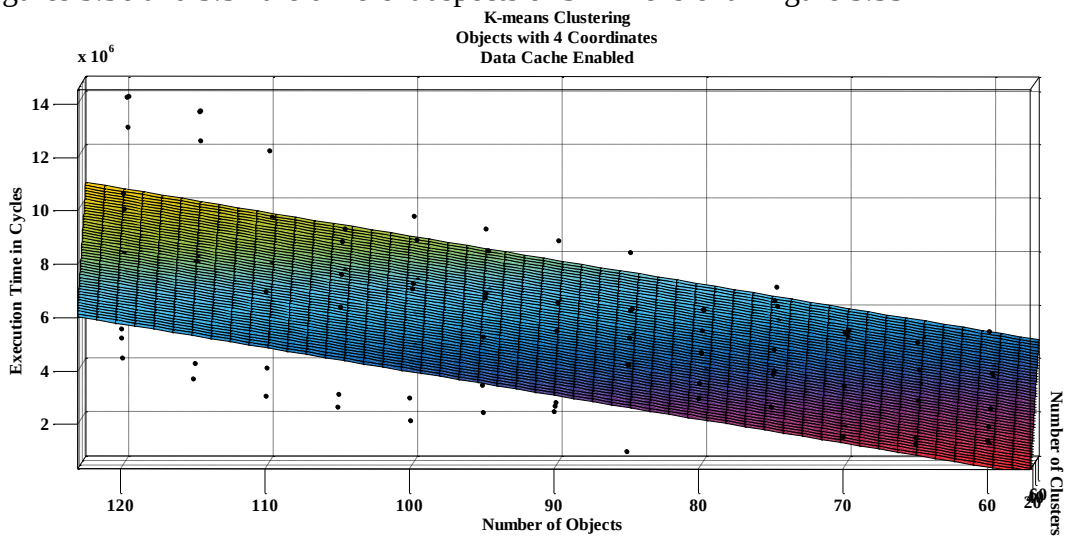


Figure 3. 36 : Different aspect of 3D Figure 3.35

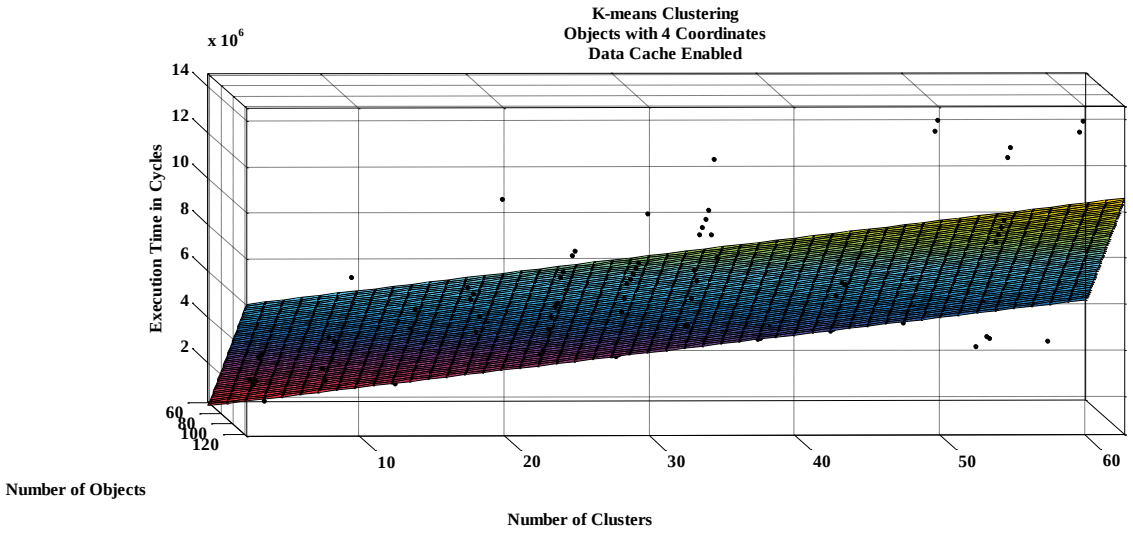


Figure 3. 37 : Different aspect of 3D Figure 3.35

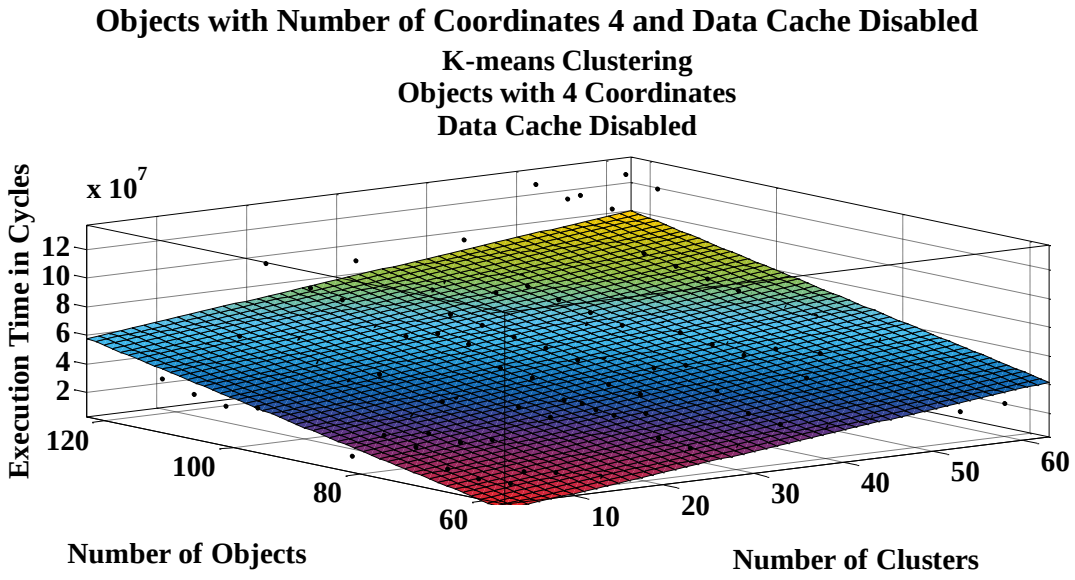


Figure 3. 38 : Execution Times for K-means Clustering for Objects with Coordinates 4, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Disabled)

Figures 3.39 and 3.39 are different aspects of 3 Dimensional Figure 3.38.

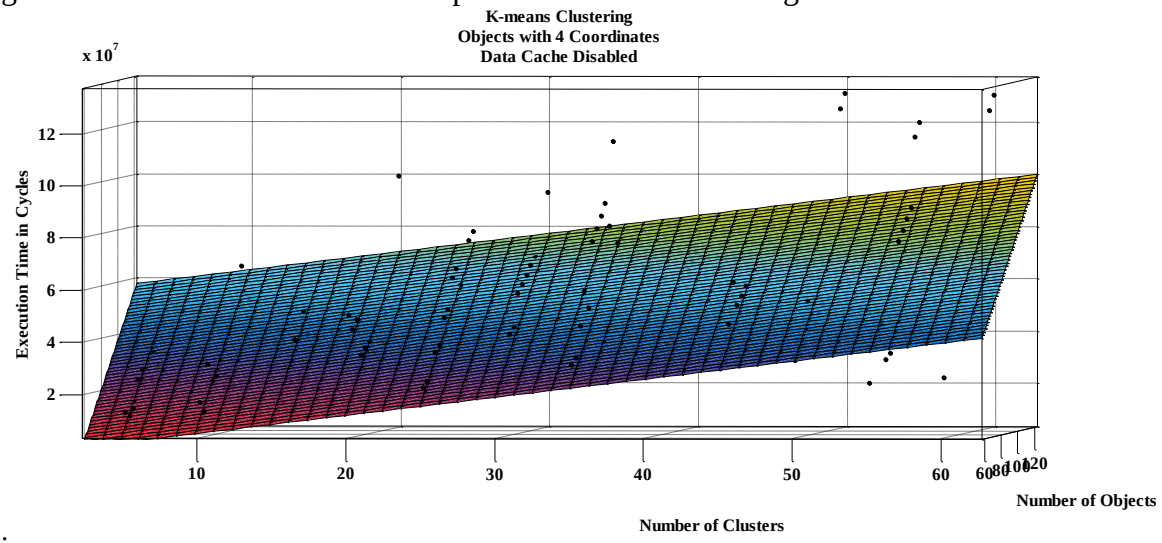


Figure 3. 39 : Different aspect of 3D Figure 3.38

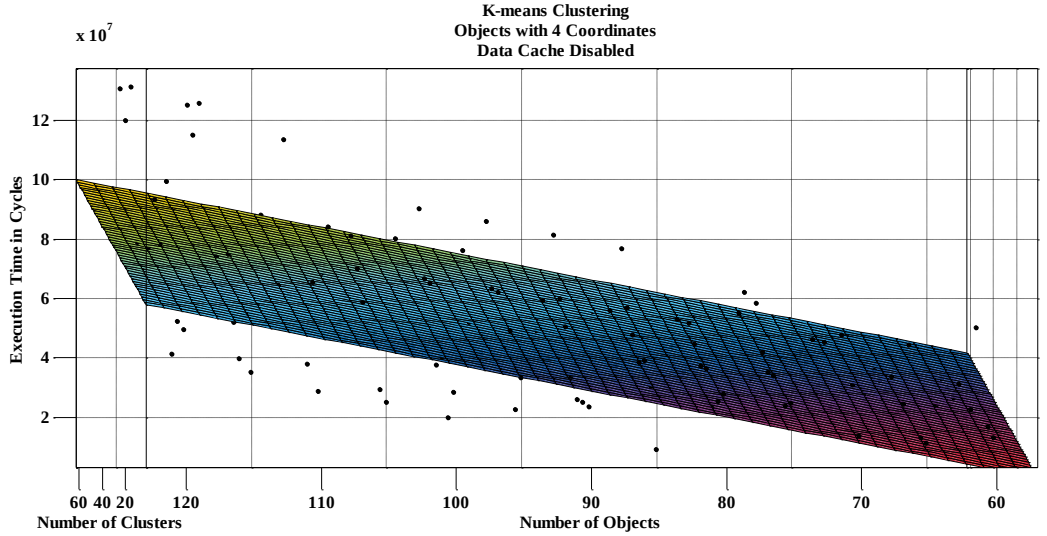


Figure 3. 40 : Different aspect of 3D Figure 3.38

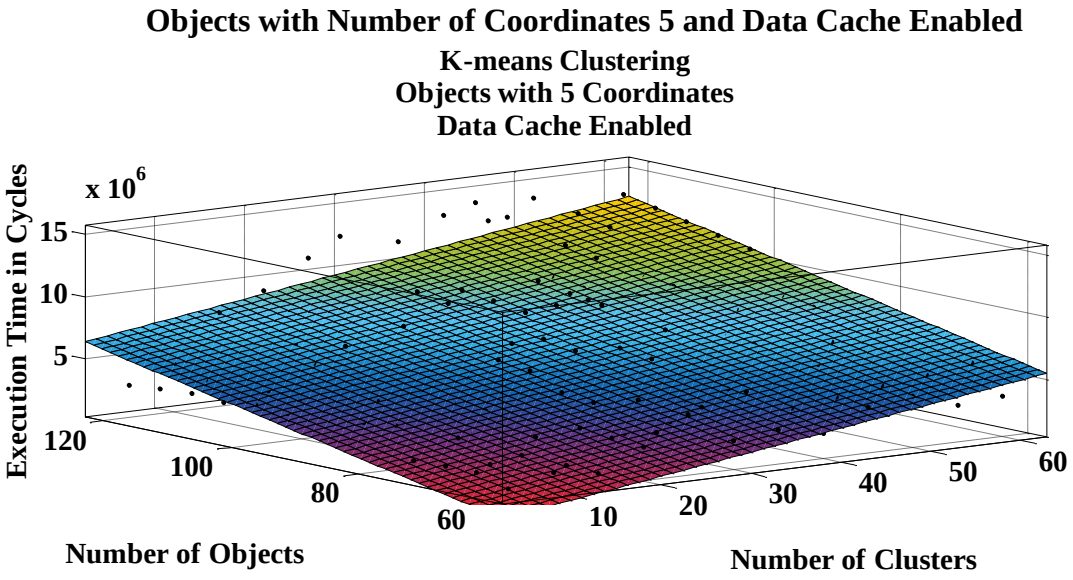


Figure 3.41 : Execution Times for K-means Clustering for Objects with Coordinates 5, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Enabled)

Figures 3.42 and 3.43 are different aspects of 3 Dimensional Figure 3.41.

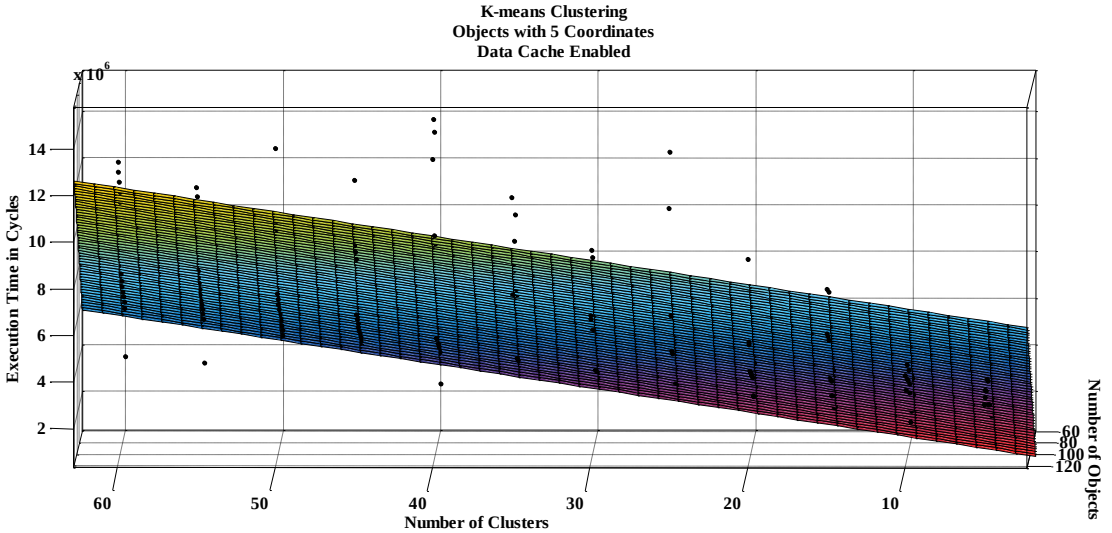


Figure 3.42 : Different aspect of 3D Figure 3.41

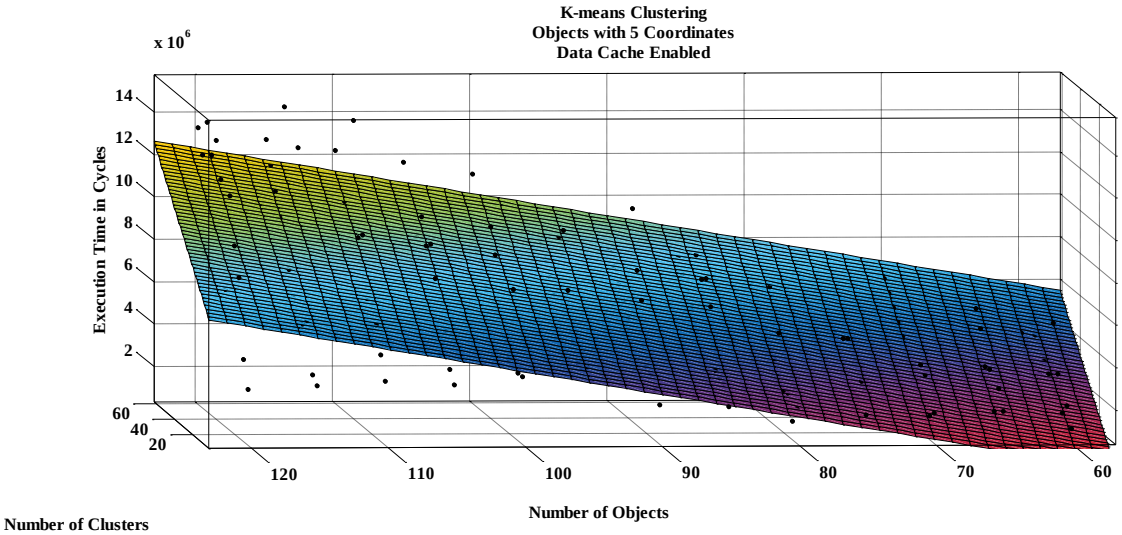


Figure 3. 43 : Different aspect of 3D Figure 3.41

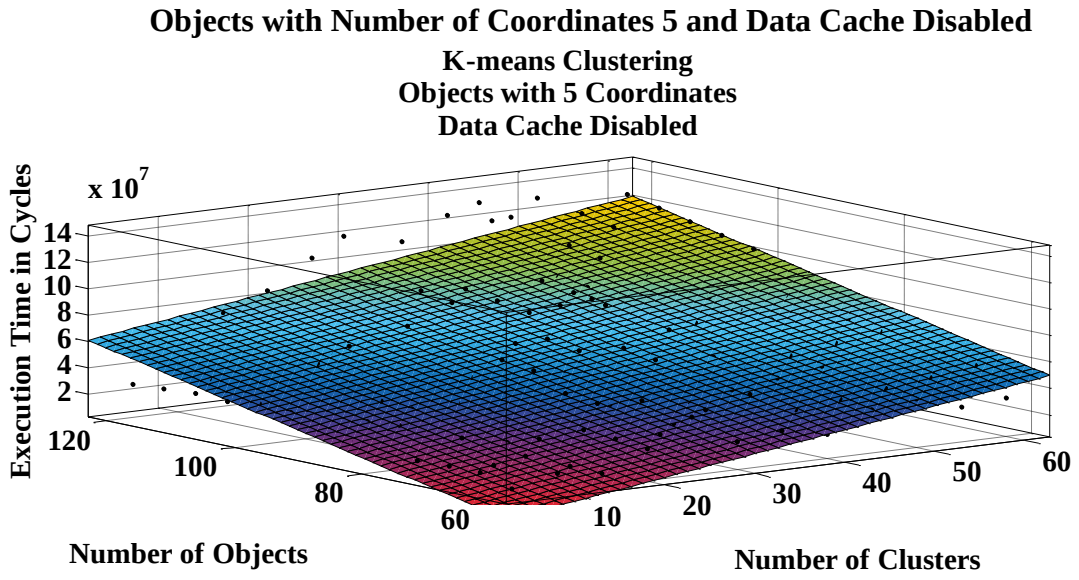


Figure 3. 44 : Execution Times for K-means Clustering for Objects with Coordinates 5, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Disabled)

Figures 3.45 and 3.46 are different aspects of 3 Dimensional Figure 3.44.

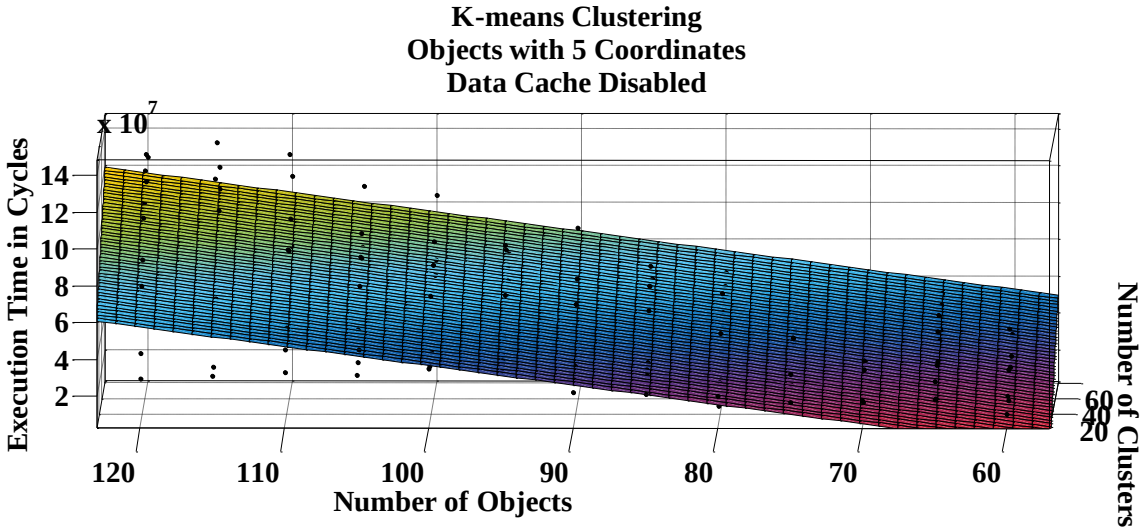


Figure 3.45 : Different aspect of 3D Figure 3.45

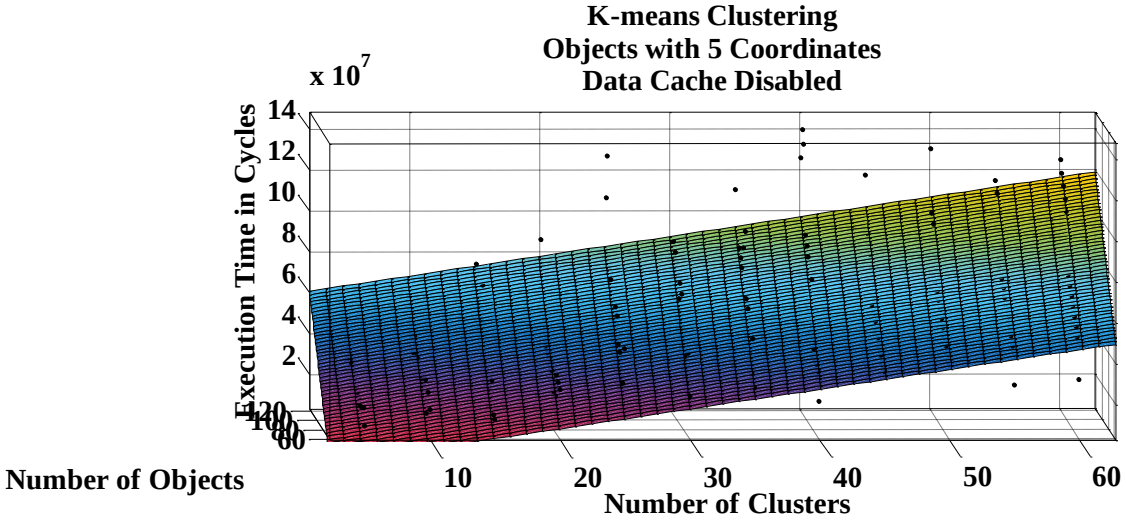


Figure 3. 46 : Different aspect of 3D Figure 3.44

Chapter 4 - Workload time prediction

In this section the methodology that was followed in order to have a reliable estimation of execution time of each kernel will be described. Each kernel requires data as input and returns the computational results for each input data set. To estimate the execution time of each kernel is a very important procedure, especially for real time embedded systems. Furthermore it is also important to have tight upper bound on execution time for a variety of reasons.

It is essential to estimate execution time for each kernel, for scheduling reasons [23], for efficient use of resources [24] or even for allowing an embedded system to identify anomalous software behaviors in order to provide online reconfiguration and re-execution in run-time [22].

Before continuing to the description of the methodology followed, important definitions will be provided about the WCET (Worst Case Estimation Time), the Regression Analysis and the Coefficient determination known as R-Square.

Worst Case Execution Time

According to Wikipedia (https://en.wikipedia.org/wiki/Worst-case_execution_time last accessed 15/10/2015) the worst-case execution time (WCET) of a computational task is the maximum length of time the task could take to execute on a particular hardware platform. It is typically used in reliable real-time systems, where understanding the worst case timing behavior of software is important for reliability or correct functional behavior.

Wilhelm et al. [46] in their work define that the determination of upper bounds on execution times is commonly called Worst-Case Execution Times (WCETs) and that it is a necessary step in the development and validation process for hard real-time systems. This problem is hard if the processor's architecture has components such as caches, pipelines, branch prediction, and other speculative components. In their paper they describe different approaches to this problem and surveys several available tools and research prototypes.

Hard real-time systems need to satisfy strict timing constraints, which stem from the systems they control. For example, a computer system that controls the behavior of an engine in a vehicle should respond to inputs within a specific amount of time. In general, upper bounds on the execution times should comply with these constraints. Unfortunately, it is not possible in general to obtain upper bounds on execution times for programs. Otherwise, someone could solve the known "halting problem" [46].

While WCET is potentially applicable to real-time systems, in practice the assurance of WCET is mainly used by real-time systems that are related to high reliability or safety. The increasing use of software in automotive systems leads to the need of using WCET analysis of software.

The Common methods used for Finding WCET are:

- End-to-end measurements of code for a subset of data inputs – test cases. This method is often called dynamic timing analysis.
- Manual static program analysis techniques such as measurements of parts of the task instructions for each function, loop, etc. and then combining them to give better predictions.

But according to Wilhelm et al. [46] these methods don't guarantee to give bounds on the execution time and so are not safe for hard real-time systems.

The worst case execution time (WCET) analysis refers to the execution time of single thread, task or process. However, on modern hardware with multi-core architecture, tasks in the system will affect the WCET of a given task if they share cache memory and other hardware features. Further, task scheduling events like blocking or interrupting should be considered in WCET analysis if they can occur in a particular system. Therefore, it is important to consider the framework in which WCET analysis is applied [46].

Regression analysis

According to the Wikipedia (https://en.wikipedia.org/wiki/Regression_analysis last accessed 15/10/2015) in statistics, regression analysis is a process, for estimating the relationships between variables. It includes techniques for modeling and analyzing several variables, when the aim is on the relationship among a dependent variable and one or more independent variables. More specifically, regression analysis helps someone understand how the value of the dependent variable changes when any one of the independent variables changes, while the independent variables stay fixed.
Sykes [51] mentions that regression techniques have long been central to the field of economic statistics. Increasingly, they have become important to lawyers and legal policy makers as well, but we can see that are also used in the field of computer science.

A wide range of test cases are found in the literature that use regression analysis for modeling in computer science, like the paper of Yang et al. "*Estimation of Execution times on*

Heterogeneous Supercomputer Architectures" [48] or the paper of Giusto et al. "*Reliable estimation of execution time of embedded software*" [49] that are modeling execution times of software on different architectures. In the work of Li & John "*Run-time Modeling and Estimation of Operating System Power Consumption*" [50] are modeling the power consumption of different Operating System tasks.

Coefficient of determination

In the Wikipedia (https://en.wikipedia.org/wiki/Coefficient_of_determination last accessed 15/10/2015) the definition Coefficient of determination in statistics is denoted R^2 or r^2. It is a number that indicates how well data fit a statistical model like a line or a curve. It is used in the context of statistical models that aim to the prediction of future outcomes or the testing of hypotheses, based on related information. It provides a measure of how well observed outcomes are replicated by the model, as the proportion of total variation of outcomes explained by the model. The coefficient of determination ranges from 0 to 1. But In some special cases R^2 can yield negative values, cases that don't concern this paper.

Modeling kernel for WCET

Since it is impractical to compute for every possible input data set a kernel's actual execution time was decided to resolve this problem by trying to identify each program's worst-case input data set or executing the kernels with a wide range of input data set. The execution times were collected and processed in order to model each kernel for the WCET for a given data set.

Regression analysis was performed using MATLAB Curve Fitting Tool in order to create the mathematical model for each kernel and finally a function was implemented in C language that computes the estimated WCET of each kernel provided independent input data sets.

Next in this section, for each kernel the mathematical model created will be presented. For each kernel and for the same input data set the results were collected for the cases of enabled and disabled data caching.

The Figures presented in Chapter 3 will also be presented and commented in this section in order to avoid changing of pages while reading this paper.

BasicMaths Cubic Solving

In this benchmark the execution time is depended on the number of the values that solve the third degree polynomial and the execution time is between 400000-600000 cycles when

caching is disabled and 120000-200000 cycles when caching is enabled. So the function returns the maximum value of the data in order to get the estimated WCET. For testing purposes the function that returns the execution time, for testing purposes, has the option to return the minimum and average value of the data collected.

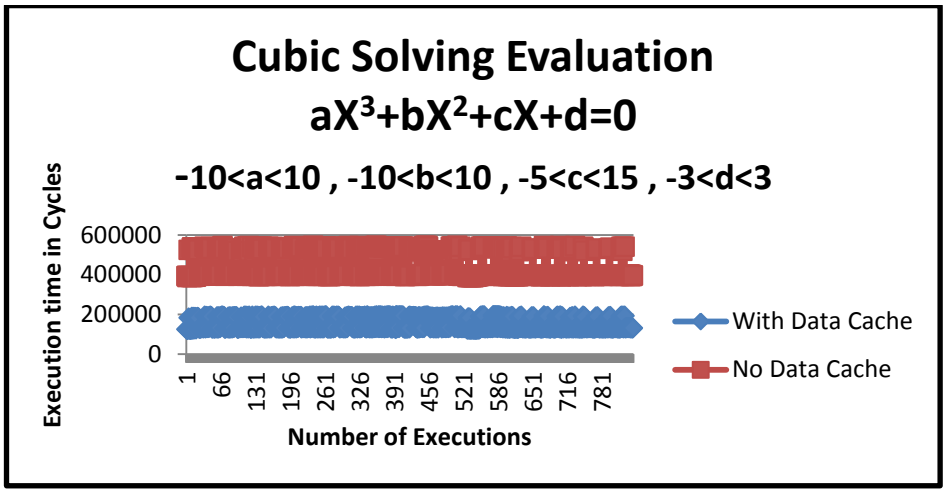


Figure 4.1 : Execution times required for solving the equation $aX^3+bX^2+cX+d=0$ with constants a,b,c,d within the ranges -10<a<10 , -10<b<10 , -5<c<15 , -3<d<3

Unsigned Integer Square Roots Computation

In this benchmark the execution time is depended on the integer that the Square Root is computed whose maximum value is around 210000 cycles when caching is disabled and around 2000 cycles when caching is enabled. So the function returns the maximum value of the data in order to get the estimated WCET. For testing purposes the function that returns the execution time, has the option to return the minimum and average value of the data collected.

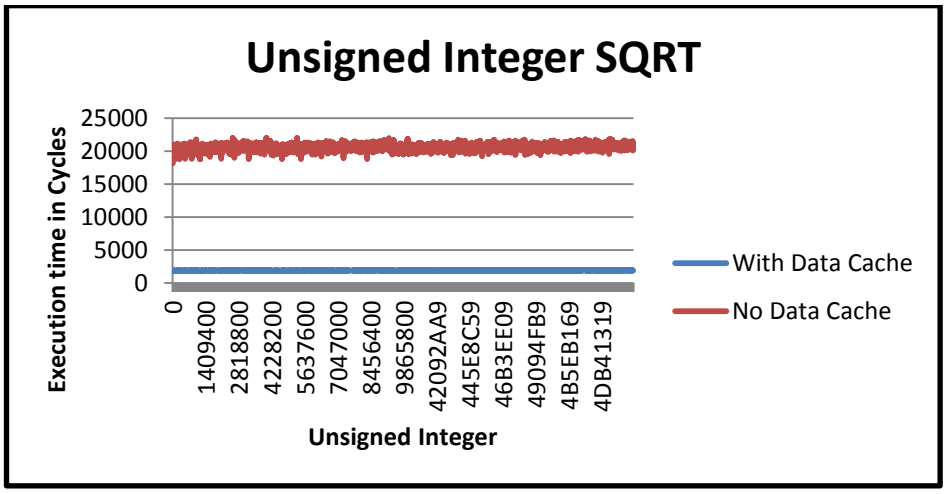


Figure 4.2 : Execution times for computing the SQRT of Unsigned Integer

Degrees to Radians Conversion

In this benchmark the execution time is independent of the value that is Converted to Degrees. As we can see in the graph the mean value is around 7800 cycles when caching is disabled and around 2800 cycles when caching is enabled. So the function returns the maximum value of the data in order to get the WCET. For testing purposes the function that returns the execution time, has the option to return the minimum and average value of the data collected.

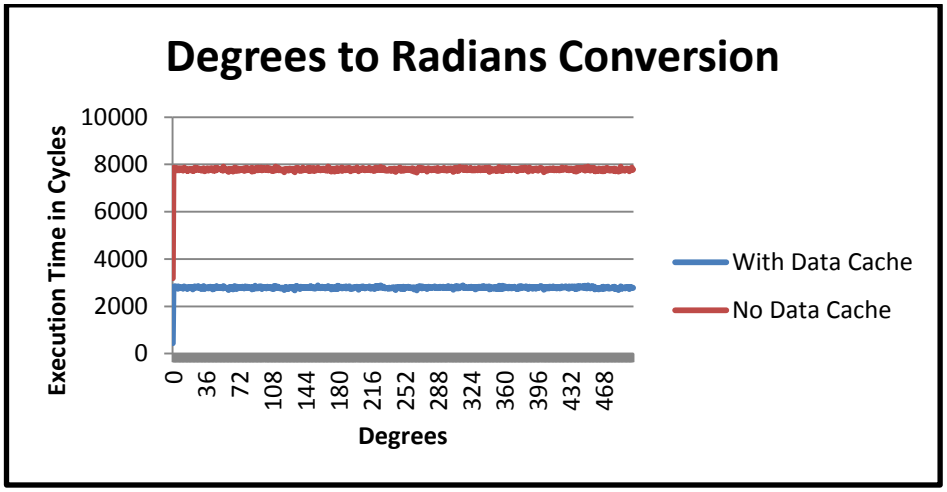


Figure 4.3 : Execution Times for the conversion of Degrees to Radians Degrees Range within

Black & Scholes

In this program it is observed that the execution times are depended on the Number of Days in a linear way in both cases of data caching disabled and data caching enabled.

The mathematical model is polynomial of first degree f(x)=p1*x+p2.

In the case of Data Cache Enabled p1=1084e+05 and p2= -2e+04

In the case of Data Cache Disabled p1=3054e+05 and p2= -5821e+04

The value of the R-Square is expected to be between 0 – 1. In both cases it is R-Square=0,999, which means that the model computed is very close to the real values collected.

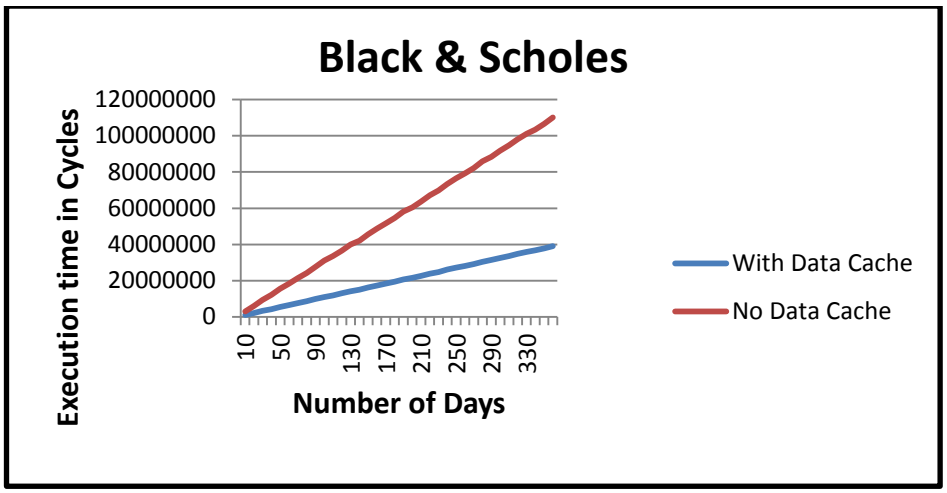


Figure 4.4 : Execution Times for the computation of a stock price for a range of days within 10 – 365

CRC32

In this program it is observed that the execution times are depended on the Number of Bytes that are used to compute the CRC32 in a linear way in both cases of data caching disabled and data caching enabled.

The mathematical model is polynomial of first degree f(x)=p1*x+p2.

In the case of Data Cache Enabled p1=42,21 and p2= 702,1

In the case of Data Cache Disabled p1= 380 and p2= 278

The value of the R-Square is expected to be between 0 – 1. In both cases it is R-Square=1. Which means that the model computed perfectly fits the data.

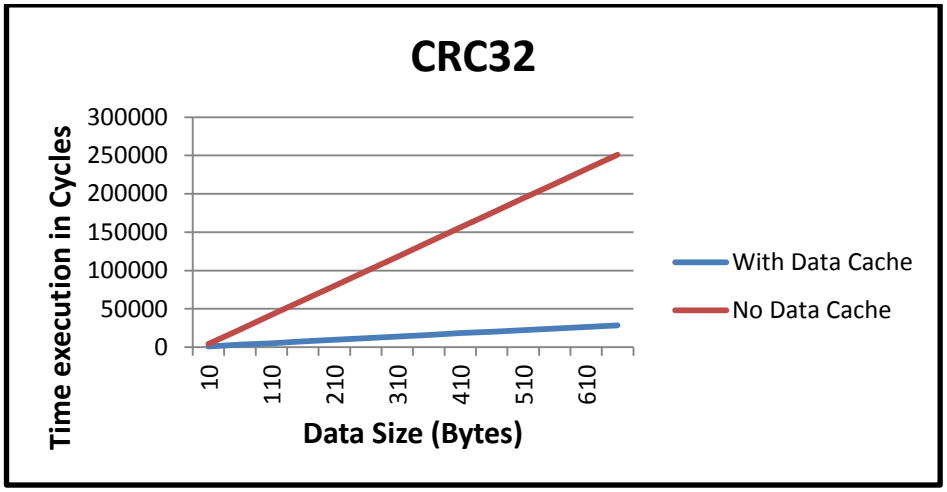


Figure 4.5 : Execution Times for creating a CRC code for a range of text size within 10-660 bytes

Dijsktra

In this program the execution times are dependent on the number of nodes that are included in the graph. As the number of Nodes increases we observe that the execution time increases in an exponential way. The model for this case is a power of first degree f(x)=p1*X^p2.

In the case of Data Cache Enabled p1=281 and p2= 2,147 with R-Square=0,9877

In the case of Data Cache Disabled p1= 2620 and p2= 2,107 with R-Square=0,9991

The data were collected with constrain that 5 shortest paths should be returned.

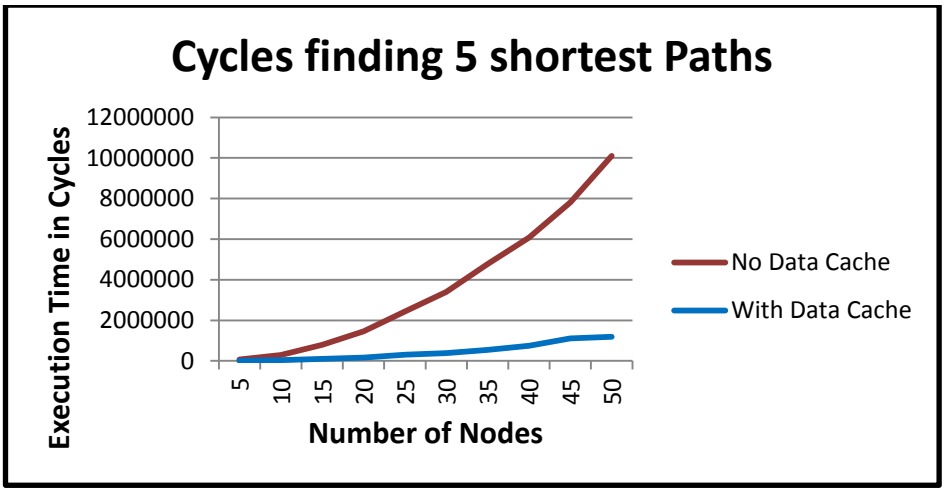


Figure 4.6 : Execution Times for finding 5 shortest paths in a Graph of adjacent Nodes within the range of 5 - 50

In this case, it must be highlighted that the number of shortest paths that are returned from the program does not affect the execution times. This can be seen from the following graph were the execution time remains almost the same while the number of required shortest paths increases and the number of nodes remains 50.

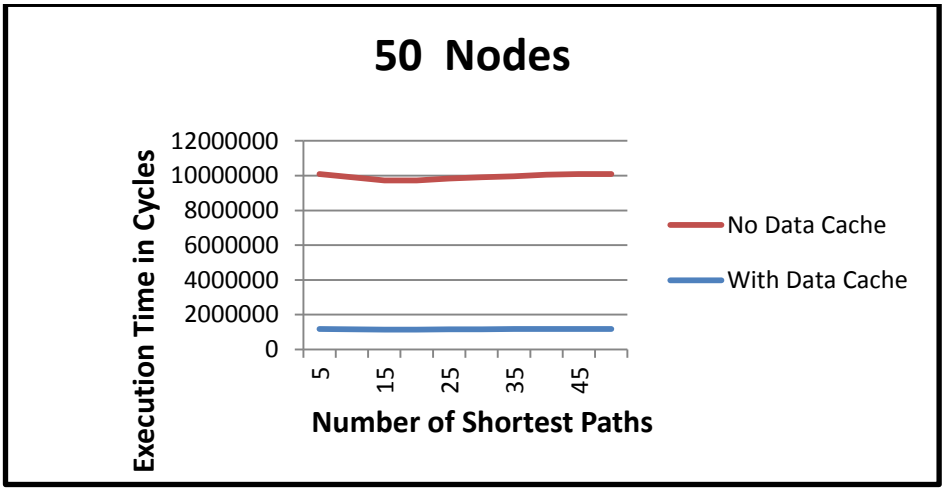


Figure 4.7 : Execution Times to find 5 – 50 shortest paths in a Graph of 50 adjacent Nodes

SHA-1

In this program it is observed that the execution times are depended on the Number of Bytes that are used to compute the SHA-1 in a linear way in both cases of data caching disabled and data caching enabled.

The mathematical model is polynomial of first degree f(x)=p1*x+p2.

In the case of Data Cache Enabled p1=276,4 and p2= -6018

In the case of Data Cache Disabled p1= 2792 and p2= -5,651e+04

The value of the R-Square is between 0 – 1 and in both cases R-Square=1. Which means that the model computed perfectly fits the data.

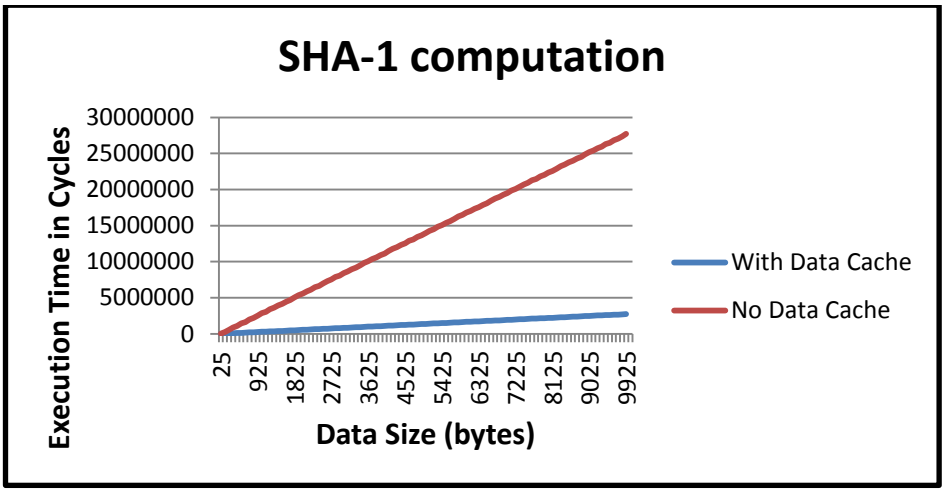


Figure 4.8: Execution Times to execute the hash function for a range of bytes within 25- 9925 bytes

FFT & IFFT

In this kernel the execution time is dependent from the next higher power of two (2), called NFFT complex number, which is greater than the Number of the signal samples. For example if the number of signal samples (Nx) is 10 then the next higher power of two (2) is NFFT complex number is 2^4=16 . In the same way if the samples of the signal samples (Nx) are 30 then the NFFT complex number is 2^5=32.

As it can be observed from the Figure 4.9 the model looks like a stairs line depended on the NFFT. In order to convert this model to linear model the lower values of the data values were excluded.

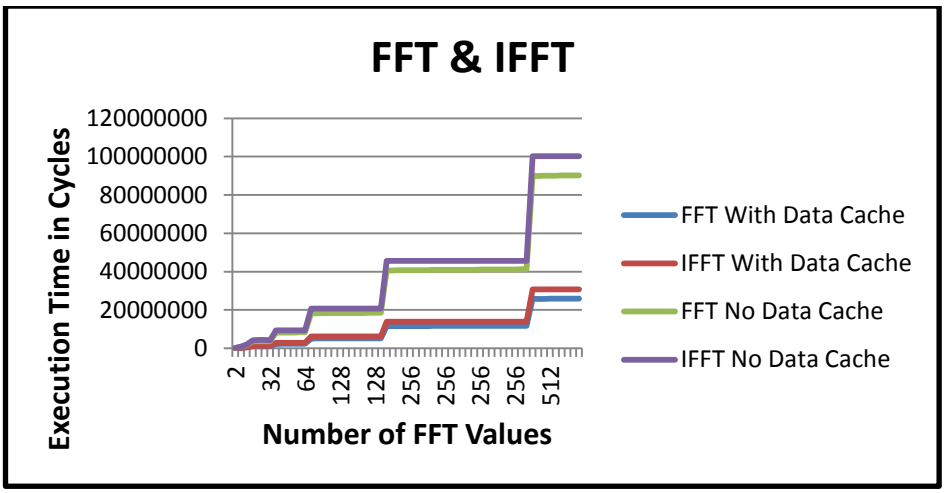


Figure 4.9 : Execution Times for computing FFT & IFFT values for a range within 2 – 512

The model from the regression analysis for this case is a power of first degree f(x)=p1*X^p2.

In the case FFT with Data Cache Enabled p1= 1.895e+04 and p2= 1,157 with R-Square=0,9999

In the case FFT with Data Cache Disabled p1= 7.275e+04 and p2= 1,142 with R-Square=1

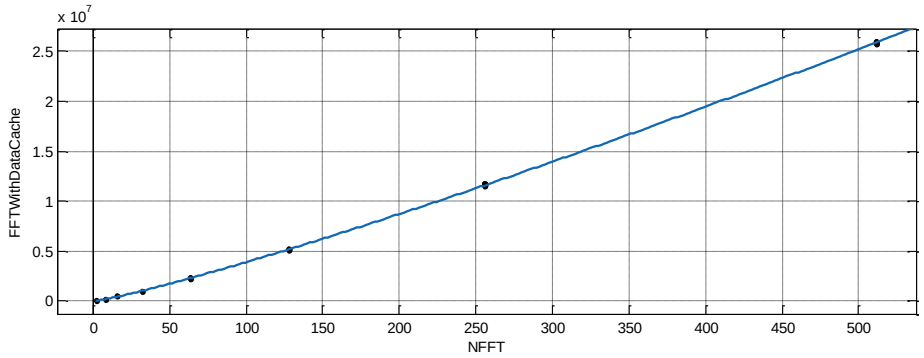


Figure 4.10 : WCET for computing FFT values for a range within 2 – 512 (Caching Enabled)

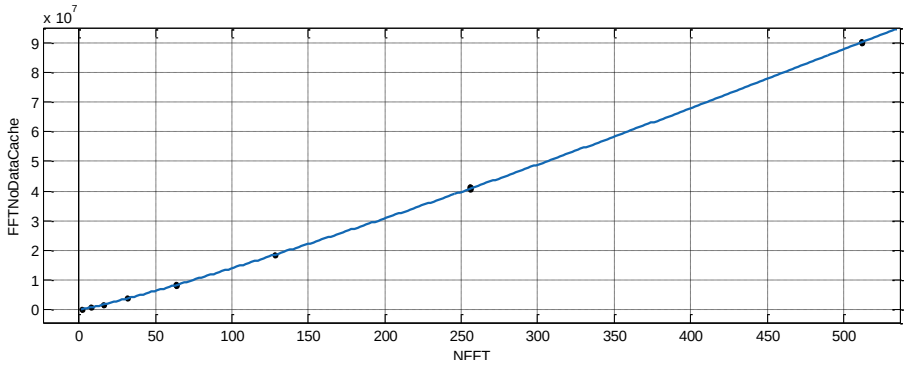


Figure 4. 11 : WCET for computing FFT values for a range within 2 – 512 (Caching Disabled)

As it concerns the regression analysis for this IFFT is also a power of first degree f(x)=p1*X^p2.

In the case IFFT with Data Cache Enabled p1= 2.386e+04 and p2= 1,148 with R-Square=1

In the case IFFT with Data Cache Disabled p1= 2.386e+04and p2= 1,148 with R-Square=1

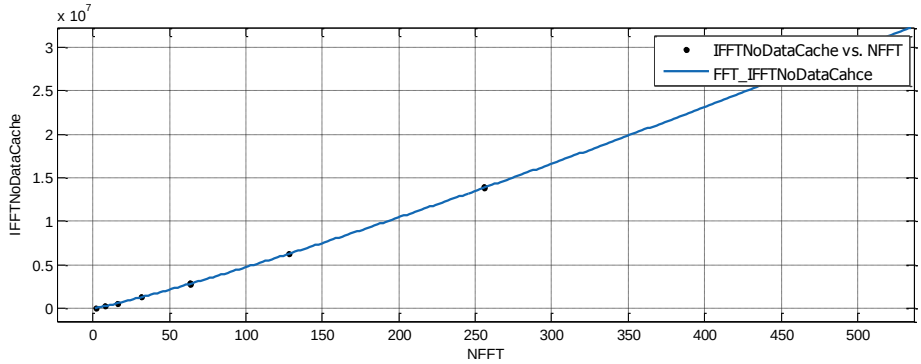


Figure 4. 12 : WCET for computing IFFT values for a range within 2 – 512 (Caching Disabled)

Image Processing – Image Denoise

In this kernel the execution time is dependent on the number of pixels of processed image. While executing the tests to collect data we confronted some problems especially in the case of caching disabled. This is the reason why it is observed in three cases that the execution times are 0. The problems arise probably due to shortage of memory. On the contrary when Data Cache was enabled the tests were executed normally and the results are presented in the graph. Despite the problems, the model was created for further tests when the configuration of the accelerator will change in the future work.

The model for this case is a power of first degree f(x)=p1*X^p2.

In the case of Data Cache Enabled p1= 928.4 and p2= 0,9574 with R-Square=0,9924

In the case of Data Cache Disabled p1= 5889 and p2= 1,044 with R-Square=0,9997

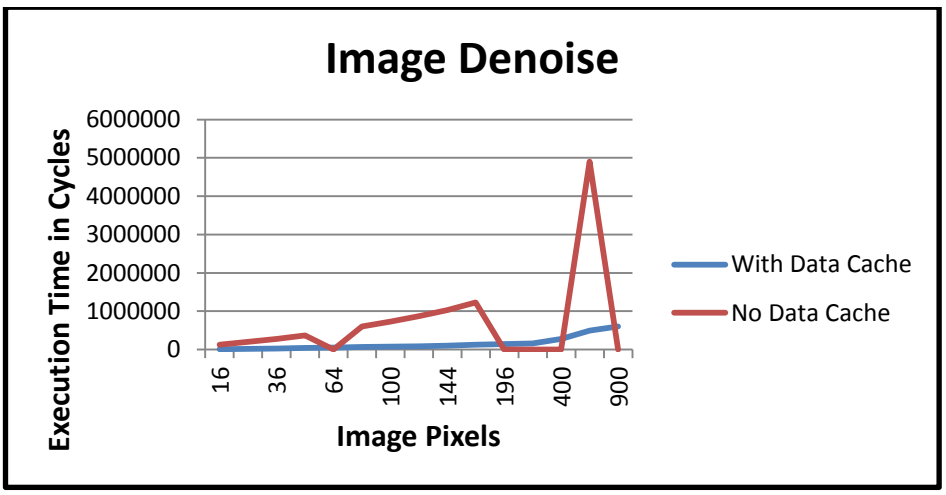


Figure 4.13 : Execution Times for applying median filter on images with image size with 16 - 900 pixels

Image Processing – Image Edge Detection

In this kernel there were some problems due to memory shortage that didn't allow a wide range of input data set. This is the reason why the image pixels range varies between 16 pixels to 196 pixels. Despite the problems, it was decided to show the results of the tests for discussion and for further examination after reconfiguration of the hardware. Apart from the above it must be highlighted that the processed images were squared images which means that they had the same number of rows and columns. This is also a future work that has to be examined after reconfiguration of the accelerator.

In the case of data cache enabled the model seems to be linear model a polynomial of first degree f(x) = p1*x + p2.

With p1 = -5.992, p2 = 4.155e+05 and Goodness of fit: R-square: 0,993

In the case of data cache disabled the model seems to be linear model exponential of second degree f(x) = a*exp(b*x) + c*exp(d*x).

With a = 4.137e+05, b = -0.001405, c = 8863, d = 0.01749 and Goodness of fit R-square= 0,9863.

But as it was mentioned at the beginning the range of input data set is too small to have a reliable model.

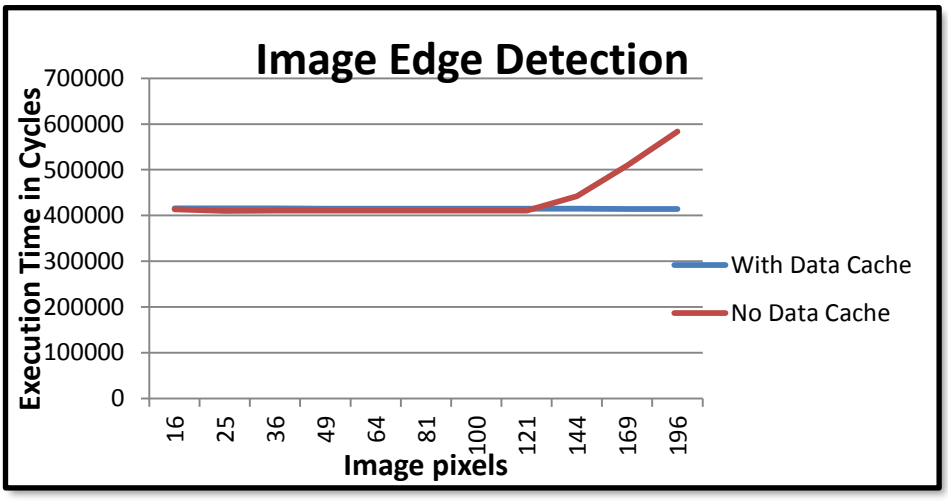


Figure 4.14 : Execution Times for applying Edge Detection on Images with image size within 16 – 196 pixels

Search String

This kernel was more complicated than the previous kernels and programs. That is due to the fact that there are two (2) independent variables. The execution times are dependent from SearchString length and the FindString length. The FindString is the string looked for (searched) in the SearchString.

In this case in order to create a reliable lineal model with the Worst Case Estimation time the lower values of the execution times were excluded and a polynomial of two variables of first degree for each variable was created.

For the case of Data Cache enabled the model is f(x,y) = p00 + p10*x + p01*y

p00 = -412,1 , p10 = 48,74 , p01 = 9,772 and Goodness of fit R-square: 0.9896

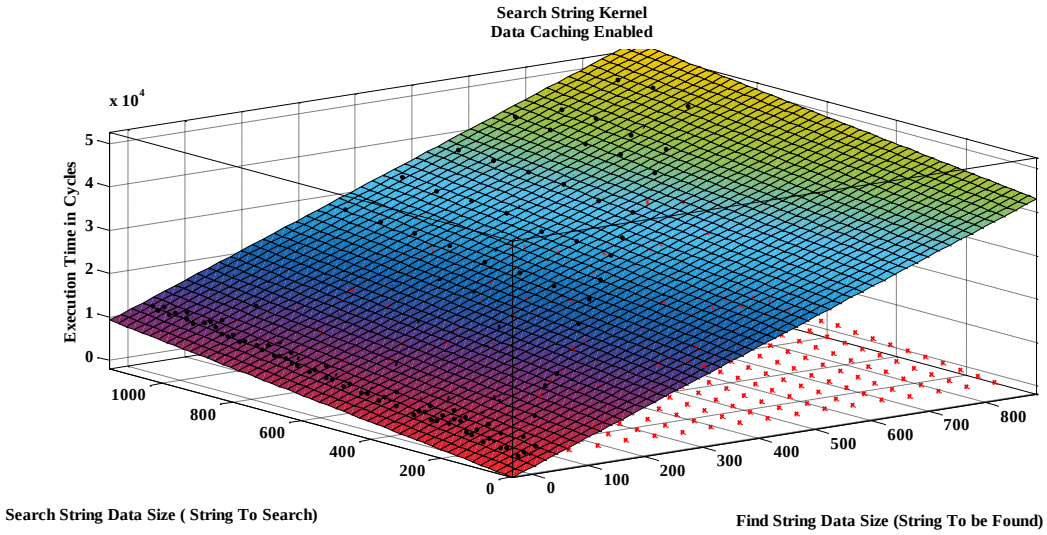


Figure 4. 15 : Execution Times for searching strings within 50-1000 chars in strings within 50-1000 chars (Caching Enabled)

For the case of Data Cache disabled the model is f(x,y) = p00 + p10*x + p01*y

p00 = 2746, p10 = 481,2 , p01 = 65,04 Goodness of fit R-square = 0,9999

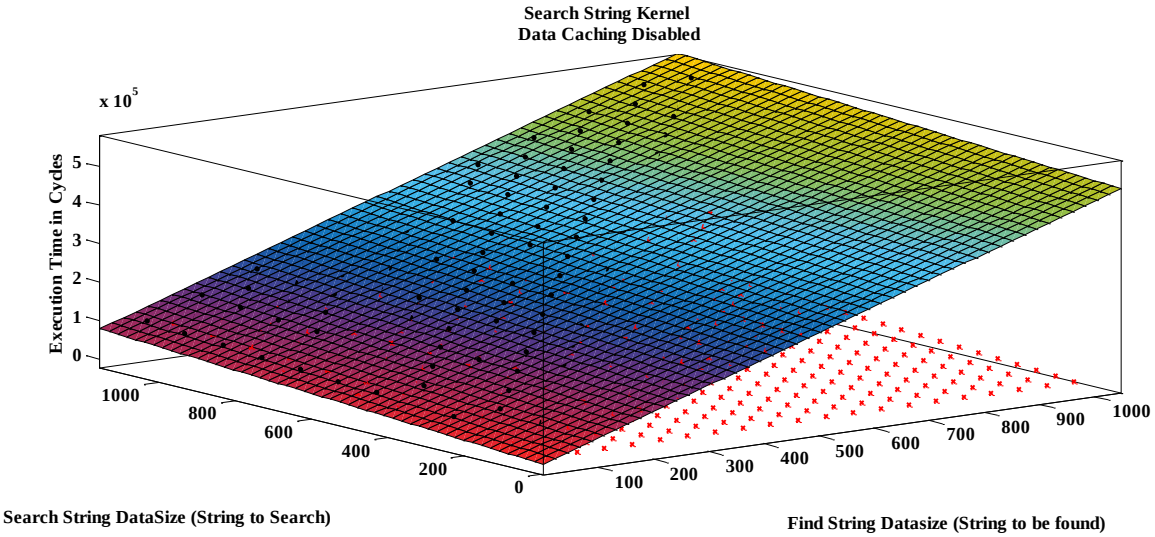


Figure 4. 16 : Execution Times for searching strings within 50-1000 chars in strings within 50-1000 chars (Caching Disabled)

K-means

This program was the most complicated of all. There are three (3) independent variables that had to be included in the model. The program is a clustering method that aims to partition **n** observations (Number of Objects 60-120) into k clusters (Number of Clusters 10-60) in which

each observation belongs to the cluster with the nearest mean. The objects have a number of Coordinates more or equal to two (2). For this case it was decided to divide the problem into sub problems. The tests were performed separately for different number of coordinates (Coordinates: 2, 3, 4, 5).

Objects with Coordinates 2

Linear model Poly11:

f(x,y) = p00 + p10*x + p01*y

Coefficients (with 95% confidence bounds):

p00 = -4.706e+06 (-5.444e+06, -3.967e+06)

p10 = 7.213e+04 (6.4e+04, 8.026e+04)

p01 = 6.311e+04 (5.561e+04, 7.061e+04)

Goodness of fit:

SSE: 1.204e+14

R-square: 0.7922

Adjusted R-square: 0.7895

RMSE: 8.872e+05

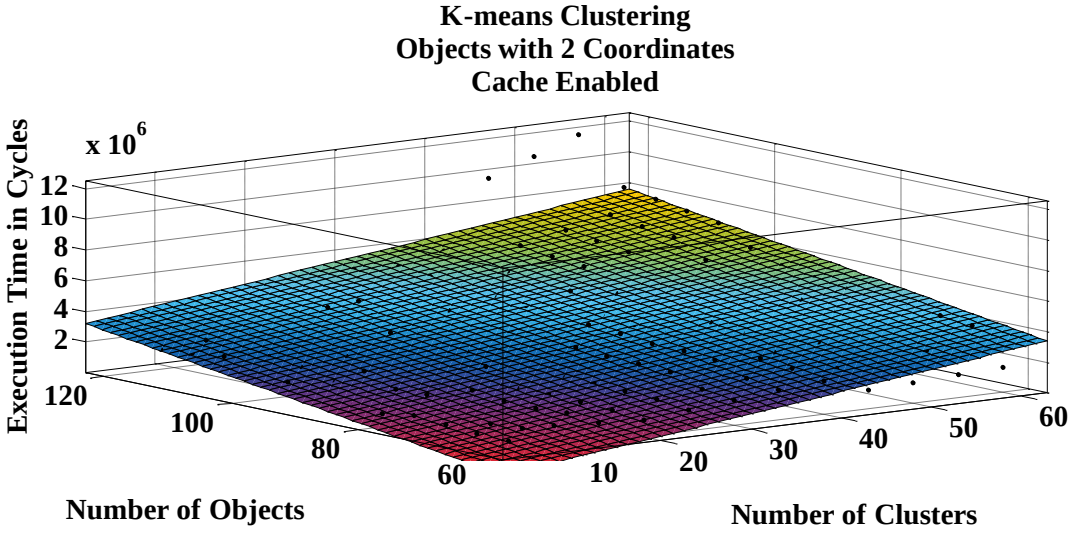


Figure 4. 17 : Execution Times for K-means Clustering for Objects with Coordinates 2 Number of Objects within 60-120 Number of Clusters 10 – 60 (Caching Enabled)

Linear model Poly11:

f(x,y) = p00 + p10*x + p01*y

Coefficients (with 95% confidence bounds):

p00 = -4.658e+07 (-5.367e+07, -3.948e+07)

p10 = 6.71e+05 (5.929e+05, 7.492e+05)

p01 = 6.231e+05 (5.51e+05, 6.952e+05)

Goodness of fit:

SSE: 1.112e+16

R-square: 0.7911
Adjusted R-square: 0.7884
RMSE: 8.526e+06

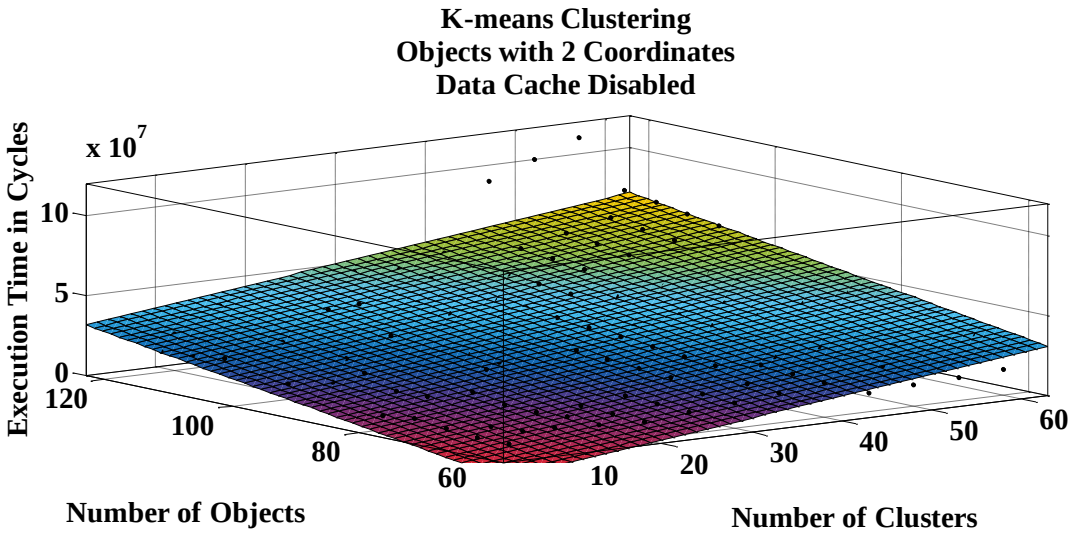


Figure 4. 18 : Execution Times for K-means Clustering for Objects with Coordinates 2, Number of Objects within 60-120, Number of Clusters 10 – 60(Caching Disabled)

Objects with Coordinates 3

Linear model Poly11:
f(x,y) = p00 + p10*x + p01*y
Coefficients (with 95% confidence bounds):
p00 = -5.592e+06 (-6.684e+06, -4.499e+06)
p10 = 8.758e+04 (7.555e+04, 9.96e+04)
p01 = 7.937e+04 (6.827e+04, 9.046e+04)

Goodness of fit:
SSE: 2.635e+14
R-square: 0.7266
Adjusted R-square: 0.7231
RMSE: 1.312e+06

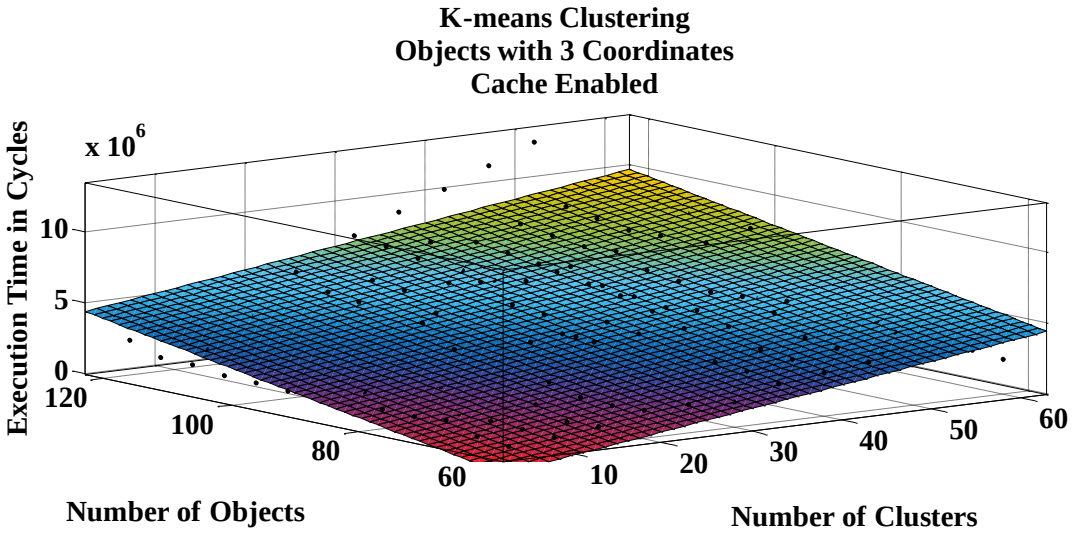


Figure 4. 19 : Execution Times for K-means Clustering for, Objects with Coordinates 3,Number of Objects within 60-120,Number of Clusters 10 – 60 (Caching Enabled)

Linear model Poly11:

f(x,y) = p00 + p10*x + p01*y

Coefficients (with 95% confidence bounds):

p00 = -5.484e+07 (-6.511e+07, -4.457e+07)

p10 = 8.089e+05 (6.959e+05, 9.22e+05)

p01 = 7.71e+05 (6.667e+05, 8.753e+05)

Goodness of fit:

SSE: 2.329e+16

R-square: 0.7296

Adjusted R-square: 0.7261

RMSE: 1.234e+07

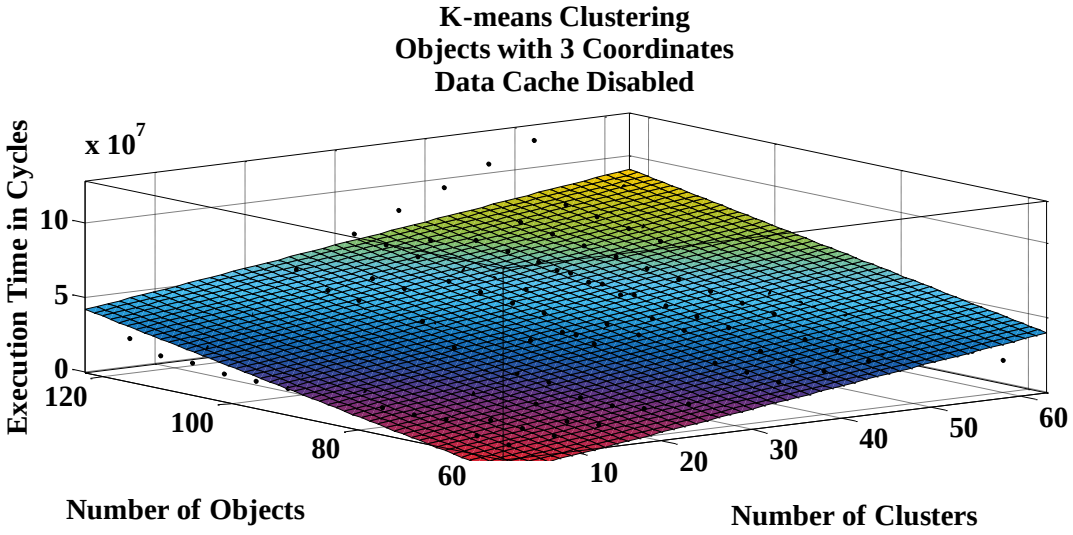


Figure 4. 20 : Execution Times for K-means Clustering for Objects with Coordinates 3, Number of Objects within 60-120, Number of Clusters 10 – 60 (Caching Disabled)

Objects with Coordinates 4

Linear model Poly11:

f(x,y) = p00 + p10*x + p01*y

Coefficients (with 95% confidence bounds):

p00 = -5.113e+06 (-6.405e+06, -3.821e+06)

p10 = 7.492e+04 (6.069e+04, 8.915e+04)

p01 = 8.934e+04 (7.621e+04, 1.025e+05)

Goodness of fit:

SSE: 3.689e+14

R-square: 0.6538

Adjusted R-square: 0.6493

RMSE: 1.553e+06

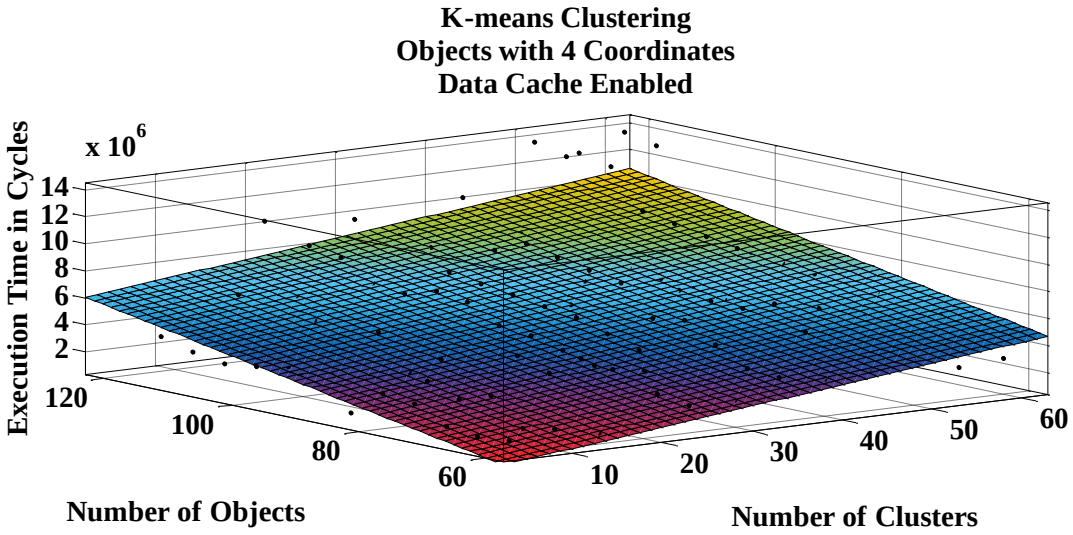


Figure 4. 21: Execution Times for K-means Clustering for Objects with Coordinates 4, Number of Objects within 60-120, Number of Clusters 10 – 60(Caching Enabled)

Linear model Poly11:

f(x,y) = p00 + p10*x + p01*y

Coefficients (with 95% confidence bounds):

p00 = -5.221e+07 (-6.434e+07, -4.009e+07)

p10 = 6.958e+05 (5.623e+05, 8.293e+05)

p01 = 8.837e+05 (7.605e+05, 1.007e+06)

Goodness of fit:

SSE: 3.247e+16

R-square: 0.6673

Adjusted R-square: 0.663

RMSE: 1.457e+07

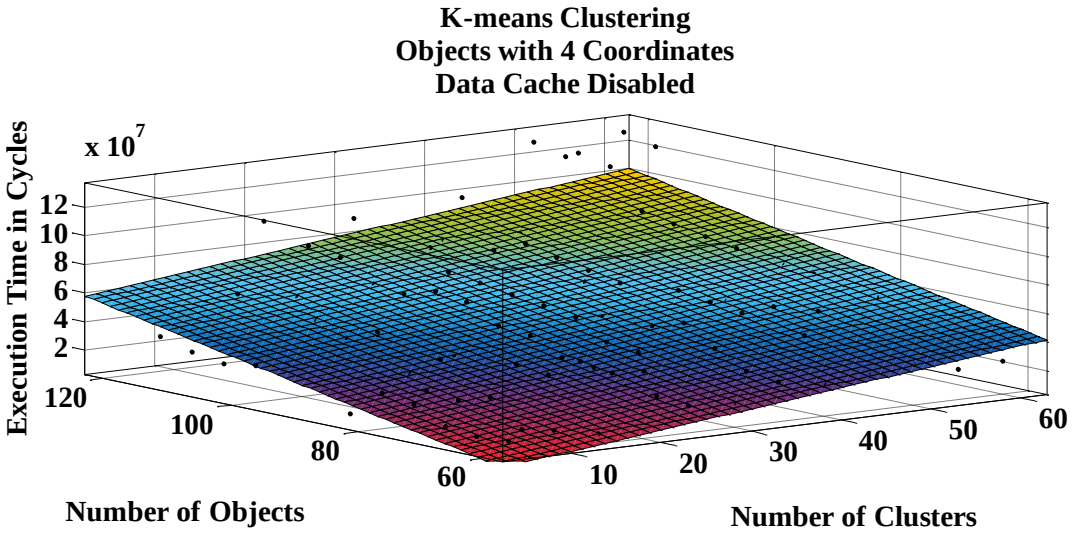


Figure 4. 22 : Execution Times for K-means Clustering for Objects with Coordinates 4,Number of Objects within 60-120,Number of Clusters 10 – 60(Caching Disabled)

Objects with Coordinates 5

Linear model Poly11:

f(x,y) = p00 + p10*x + p01*y

Coefficients (with 95% confidence bounds):

p00 = -7.207e+06 (-8.568e+06, -5.845e+06)

p10 = 1.037e+05 (8.87e+04, 1.187e+05)

p01 = 1.084e+05 (9.46e+04, 1.223e+05)

Goodness of fit:

SSE: 4.093e+14

R-square: 0.7361

Adjusted R-square: 0.7326

RMSE: 1.636e+06

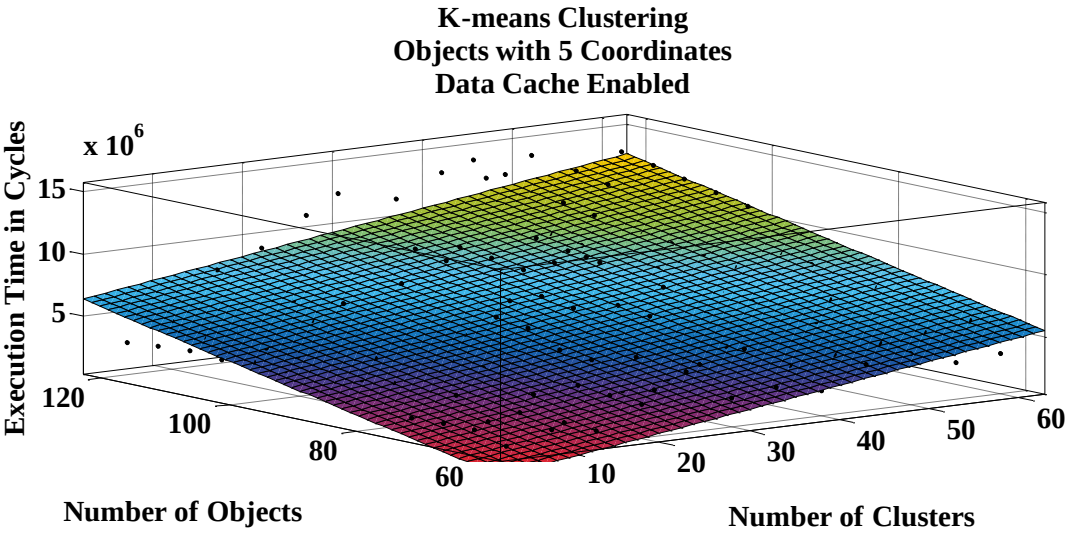


Figure 4. 23 : Execution Times for K-means Clustering for Objects with Coordinates 5, Number of Objects within 60-120, Number of Clusters 10 – 60(Caching Enabled)

Linear model Poly11:

f(x,y) = p00 + p10*x + p01*y

Coefficients (with 95% confidence bounds):

p00 = -7.023e+07 (-8.299e+07, -5.748e+07)

p10 = 9.584e+05 (8.18e+05, 1.099e+06)

p01 = 1.048e+06 (9.186e+05, 1.178e+06)

Goodness of fit:

SSE: 3.593e+16

R-square: 0.7408

Adjusted R-square: 0.7374

RMSE: 1.532e+07

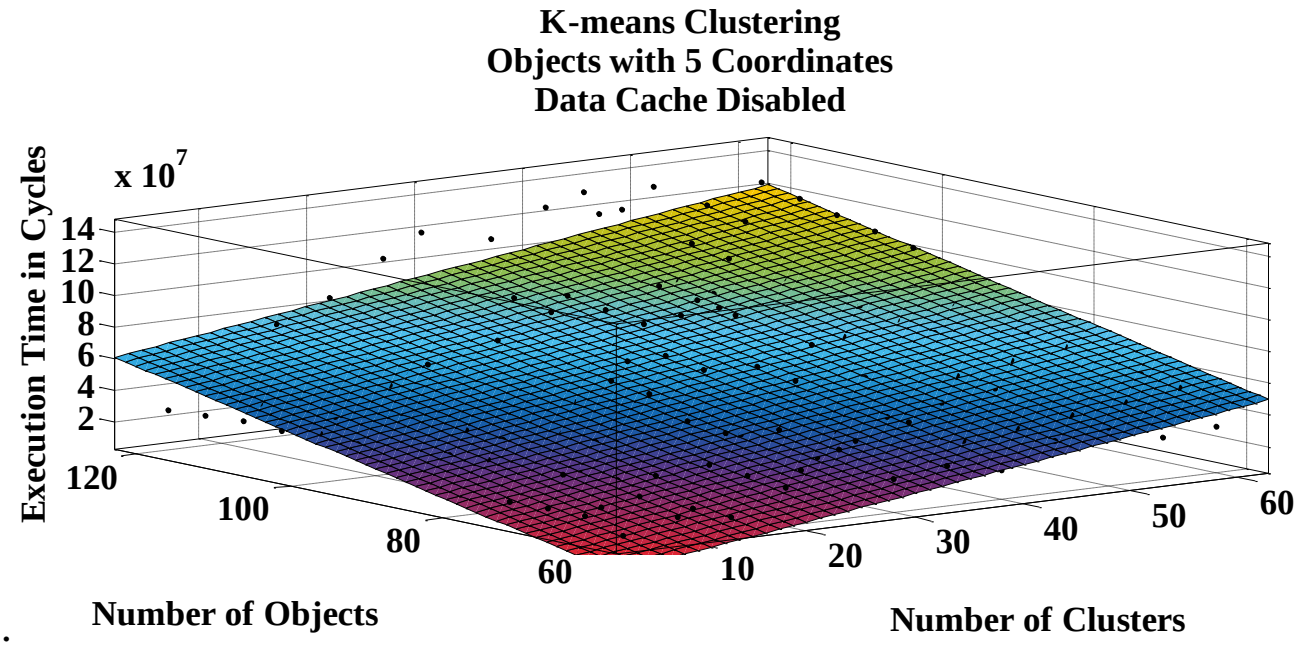


Figure 4. 24 : Execution Times for K-means Clustering for Objects with Coordinates 5 Number of Objects within 60-120 Number of Clusters 10 – 60(Caching Disabled)

Chapter 5 - Kernels Concurrent Scheduling

According to Brucker [27], since mid-fifties scheduling problems concern computer scientists as a tool to increase the performance of computer systems. Furthermore, scheduling problems have been investigated and classified according to their computational complexity. The scheduling problem nowadays has different views and a lot of research has been done in this domain. The problems are classified between single machines scheduling problems, parallel machines scheduling problems [32], for homogeneous and heterogeneous systems scheduling problems [29, 30]. There are also papers that tackle independent tasks/jobs problems [26], dependent tasks/jobs scheduling problems and due-date scheduling problems [27].
In most of the cases the scheduling problem is a Directed Acyclic Graph DAG problem whose optimal solution is considered to be NP-Hard problem. Brucker [27] in his thesis, describes a great number of scheduling algorithms found in the literature, like Sarkar, HLFET (Highest Level First with Estimated Time) , ETF (Earliest Time First), ISH (Insertion Scheduling Heuristic), FLB (Fast Load Balancing), DSC (Dominant Sequence Clustering), CASS-II, DCP (Dynamic Critical Path), MCP (Modified Critical Path) and MD that as he mentions work on homogeneous systems but are useful to be studied and also work on heterogeneous systems.

In our case we used the source code of Doss [53] which solves the multiprocessor scheduling problem using a Partition Approximation algorithm [33, 52, 53]. Doss [52] describes the partition scheduling problem as the problem that asks that a given set (S) of integer values that represent process running times, to be subdivided into two subsets S' and S" in a way that the sum of all process running times in the first set (S') is equal to the sum of all process running times in the second set (S"). In this way, the total sum of process running times in both subsets is equivalent, or very close, to the half of the total cost of all process running times in the original set (S). This is a NP-Complete problem and can be proven via a reduction from three dimensional matching.

As it is well known NP-Complete problems do not have to date, any polynomial time algorithm that can solve them. It is supposed that such problems don’t have a polynomial time solution. However, approximation algorithms deal with many NP-Complete problems, a reason why they have been developed. Approximation algorithms are algorithms which do not solve the NP-Complete problem optimally in all cases but they trade accuracy for performance. As Doss [53] highlights, not all NP-Complete problems can be approximated; however, the partition scheduling problem has several. With this technique an approximation for the partition scheduling problem can be generalized to approximate scheduling for many

processors machines where the number of processors is a power of two, like 2^1, 2^2, 2^3, 2^4, and so on [34].

Scheduling Simulation

In order to perform the simulation we assume that our platform has two (2) preloaded accelerators with the same configuration. This means that each accelerator can execute the kernels in the same time and according to the models that have been constructed previously.

In Table 5.1 there is a list of seven (7) kernels that must be executed in parallel from the platform with the two (2) accelerators. In Column 3 we can see the execution time of each kernel, provided by the models. In Column 4 there is the input data set for each kernel finally, we assume that all kernels are executed with Caching Enabled.

Task ID	Kernel Name	Execution Time	Input Data Set	Caching
T1	Unsigned Integer SQRT	1899	Integer: 1000000	Yes
T2	Degrees to Radians	2793	Degrees: 180^o	Yes
T3	CRC32	21807	Data Size : 500	Yes
T4	Search String	5439	20 Find String Size 100 Search String Size	Yes
T5	Image Denoise	161933	Image Pixels: 225	Yes
T6	SHA-1	270382	Data Size : 1000	Yes
T7	Cubic Solving	154998		Yes

Table 5. 1 : Independent Task Simulation

The results of the Schedule are:

```
size [7] cpus [2]
 log2 [1]
 queue size [2]
cpu = [0]
 len = [5] { 270382 21807 5439 2793 1899 }
 sum = [302320]
cpu = [1]
 len = [2] { 161933 154998 }
 sum = [316931]
```

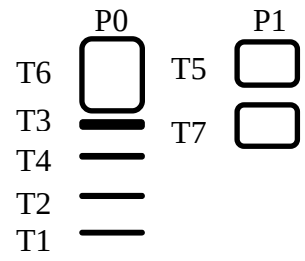
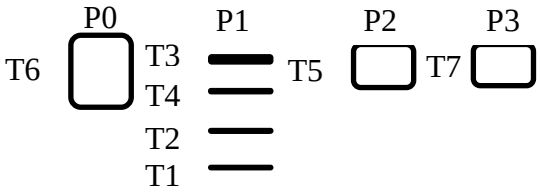


Figure 5. 1: Schedule for 7 tasks on 2 cpus

Now if there were four (4) accelerators in the platform the results would be:

```
size [7] cpus [4]
 log2 [2]
 queue size [4]
cpu = [0]
 len = [1] { 270382 }
 sum = [270382]
```



cpu = [1] len = [4] { 21807 5439 2793 1899 } sum = [31938] cpu = [2] len = [1] { 161933 } sum = [161933] cpu = [3] len = [1] { 154998 } sum = [154998]	

Figure 5. 2 : Schedule for 7 tasks on 4 cpus

Now as we can see the SHA-1 lasts for very long time comparing the other kernels execution times. Since all tasks are independent and they start at the same time, the scheduler could reduce power consumption of the platform by disabling the caching from some accelerators and manage to finish all the kernels at the same time. In this case the Table 5.1 could change to Table 5.2.

Task ID	Kernel Name	Execution Time	Input Data Set	Caching
T1	Unsigned Integer SQRT	20413	Integer: 1000000	No
T2	Degrees to Radians	7788	Degrees: 180^{o}	No
T3	CRC32	21807	Data Size : 500	Yes
T4	Search String	5439	20 Find String Size 100 Search String Size	Yes
T5	Image Denoise	161933	Image Pixels: 225	Yes
T6	SHA-1	270382	Data Size : 1000	Yes
T7	Cubic Solving	154998		Yes

Table 5. 2: Independent Tasks Simulation with power consumption optimization

And the results would be for 2 accelerators:

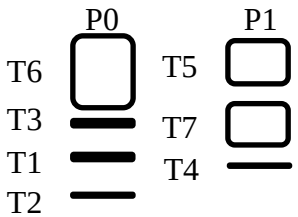
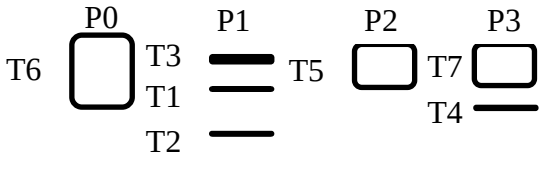
size [7] cpus [2] log2 [1] queue size [2] cpu = [0] len = [4] { 270382 21807 20413 7788 } sum = [320390] cpu = [1] len = [3] { 161933 154998 5439 } sum = [322370]	

Figure 5. 3 : Schedule for 7 tasks on 2 cpus with power optimization

The results for 4 accelerators:

size [7] cpus [4] log2 [2] queue size [4] cpu = [0] len = [1] { 270382 } sum = [270382] cpu = [1]	

len = [3] { 21807 20413 7788 } sum = [50008] cpu = [2] len = [1] { 161933 } sum = [161933] cpu = [3] len = [2] { 154998 5439 } sum = [160437]	

Figure 5. 4 : Schedule for 7 tasks on 4 cpus (With Power Optimization)

As we can see from the results all kernels would finish after 270382 cycles which is the highest execution time of SHA-1 kernel, but in the second case the power consumption would be less.

Conclusions

The main objective of this thesis was to describe the steps followed to present a methodology for retrieving the best performance in execution time and power consumption of independent tasks executed by optimally configured processing element in an embedded heterogeneous system.

The steps of the procedure / methodology are listed below.

1. Choosing kernels of different domains of embedded systems, like simple math computations, digital signal processing, image processing, network, economics, etc. Adapt the code in C to run on the Processing Elements, in our case the MicroBlaze processor in the Zynq-7000 All programmable SoC FPGA platform of Xilinx.

2. Execution of the Kernels for different data inputs to log the execution time. In order to log execution times for enabled and disabled caching tests were executed for each case. The results showed that in all kernels the execution times were higher when caching was disabled than when it was enabled according to the theory.

3. Regression analysis was performed using the execution times logged for each kernel for modeling the execution time (dependent variable) and data input (independent variables). To implement this process the Matlab Curve fitting tool was used.

4. All the models were included in a single function which returns the predicted execution time for each kernel depended on different inputs.

5. Finally, a list of tasks is imported in a multiprocessor scheduler using Partition Approximation, which would decide in what order the tasks will be executed from a specified number of processing elements.

Future Work

This thesis is a part of a bigger project; at this point there is the need to test the predicted/estimated WCET with the measured WCET of each kernel, like it is done in the paper Li et al. [24]. The ratio between the measured and the predicted WCET will show how close the predicted/estimated value is to the measured value and accordingly if it is needed to improve the models created.

As it was also mentioned in section 5.4 the configuration of the accelerator should be changed in order to achieve better results of the image processing kernel (edge detection).

Another important future work, as it was mentioned in the motivation section 1.1 is to compact all this steps to work together by implementing to the system manager the scheduling and the message passing ability to the accelerators, preload and configure them with the specific tasks.

Finally the scheduler must be modified with the intelligence of taking into account the power consumption reduction by choosing cache enabling/disabling of tasks.

Bibliography

[1] Su, L. T. (2013, February). Architecting the future through heterogeneous computing. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International* (pp. 8-11). IEEE.

[2] Brookwood, N.(2013, October). Everything You Always Wanted to Know About HAS. Available http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/10/Everything_You_Always_Wanted_to_Know_About_HSA_Final2.pdf (last accessed 10/7/2015).

[3] Kyriazis, G. (2012). Heterogeneous system architecture: A technical review. *AMD Fusion Developer Summit*.

[4] Buchty, R., Heuveline, V., Karl, W. and Weiss, J.P. (2012). *A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators.* Concurrency and Computation: Practice and Experience, vol. 24, no. 7, pages 663-675.

[5] Youenn Corre (2013). *Automated Generation of Heterogeneous Multiprocessor Architectures: Software and Hardware Aspects.* Universite de Bretagne Sud.

[6] Reagen, B., Adolf, R., Shao, Y. S., Wei, G. Y., & Brooks, D. (2014, October). MachSuite: Benchmarks for accelerator design and customized architectures. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on* (pp. 110-119). IEEE.

[7] Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., & Brown, R. B. (2001, December). MiBench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on* (pp. 3-14). IEEE.

[8] MiBench Benchmark Suite available at http://wwweb.eecs.umich.edu/mibench/ (last accessed 17/7/2015)

[9] John Burkardt open source code http://people.sc.fsu.edu/~jburkardt/c_src/c_src.html (last accesses 18/7/2015)

[10] MacBeth, J. D., & Merville, L. J. (1979). An Empirical Examination of the Black-Scholes Call Option Pricing Model. *The Journal of Finance*, *34*(5), 1173-1186.

[11] MacQueen, J. B. (1967). Some methods for classication and analysis of multivariate observations. Proceedings of the Fifth Symposium on Math, Statistics, and Probability (pp. 281{297). Berkeley, CA: University of California Press.

[12] Wei-keng Liao Kmeans software package available at http://users.eecs.northwestern.edu/~wkliao/Kmeans/index.html (last accessed 19/7/2015)

[13] Sen, R., & Wood, D. A. Cache Power Budgeting for Performance.

[14] Xilinx Zynq®-7000 documentation. Available: http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-

G-V7.pdf (last accessed 2/8/2015)

[15] Xilinx (May, 2015). *Zynq-7000 All Programmable SoC Overview DS190 (v1.8)* Available: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (last accessed 2/8/2015)

[16] Xilinx. (2014). *Microblaze soft processor core.* [Online]. Available: http://www.xilinx.com/tools/microblaze.htm(last accessed 2/8/2015)

[17] Franklin, Z. R. (2014). *Using High-level Synthesis to Predict and Preempt Attacks on Industrial Control Systems* (Doctoral dissertation, Virginia Tech).

[18] Xilinx. (2015). *MicroBlaze Processor Reference Guide*. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug984-vivado-microblaze-ref.pdf (last accessed 9/8/2015)

[19] Press, W. H. (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

[20] MHADHBI, I., Rejeb, N., OTHMEN, S. B., & SAOUD, S. B. (2014). Performance Evaluation of FPGA Soft Cores Configurations Case of Xilinx MicroBlaze.

[21] Horspool, R. N. (1980). Practical fast searching in strings. *Software: Practice and Experience*, *10*(6), 501-506.

[22] Christoforakis, I., Tomoutzoglou, O., Bakoyiannis, D., & Kornaros, G. (2014, August). Runtime Adaptation of Embedded Tasks with A-Priori Known Timing Behavior Utilizing On-Line Partner-Core Monitoring and Recovery. In *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on* (pp. 1-8). IEEE.

[23] Andrei, A., Eles, P., Peng, Z., & Rosen, J. (2008, January). Predictable implementation of real-time applications on multiprocessor systems-on-chip. In *VLSI Design, 2008. VLSID 2008. 21st International Conference on* (pp. 103-110). IEEE.

[24] Li, Y. T. S., Malik, S., & Wolfe, A. (1999). Performance estimation of embedded software with instruction cache modeling. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, *4*(3), 257-279.

[25] Zhong, J., & He, B. (2014). Kernelet: High-throughput gpu kernel executions with dynamic slicing and scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, *25*(6), 1522-1532.

[26] Bruno, J., Coffman Jr, E. G., & Sethi, R. (1974). Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, *17*(7), 382-387.

[27] Brucker, P., & Brucker, P. (2007). *Scheduling algorithms* (Vol. 3). Berlin: Springer.

[28] Forti, A. (2006). DAG Scheduling for grid computing systems.

[29] Sakellariou, R., & Zhao, H. (2004, April). A hybrid heuristic for DAG scheduling on heterogeneous systems. In *Parallel and Distributed Processing*

Symposium, 2004. Proceedings. 18th International (p. 111). IEEE.

[30] Hall, B. (2005). *Slot Scheduling: General-Purpose Multiprocessor Scheduling for Heterogeneous Workloads*. Computer Science Department, University of Texas at Austin.

[31] Lam, Y. M. (2012). Integrated task clustering, mapping and scheduling for heterogeneous computing systems. *Int J Comput Sci Inform Tech*, *4*(1), 275-80.

[32] Webster, S., & Azizoglu, M. (2001). Dynamic programming algorithms for scheduling parallel machines with family setup times. *Computers & Operations Research*, *28*(2), 127-137.

[33] Mertens, S. (2003). The Easiest Hard Problem: Number Partitioning. (Last accessed 2/10/2015 from http://www.arxiv.org/pdf/cond-mat/0310317).

[34] Johnson, D. S. (1985). The NP-completeness column: an ongoing guide. *Journal of Algorithms*, *6*(3), 434-451.

[35] Wiangtong, T., Cheung, P. Y., & Luk, W. (2005). Hardware/software codesign: a systematic approach targeting data-intensive applications. *Signal Processing Magazine, IEEE*, *22*(3), 14-22.

[36] Varbanescu, A. L., Hijma, P., Van Nieuwpoort, R., & Bal, H. (2011, May). Towards an effective unified programming model for many-cores. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on* (pp. 681-692). IEEE.

[37] Che, S., Li, J., Sheaffer, J. W., Skadron, K., & Lach, J. (2008, June). Accelerating compute-intensive applications with GPUs and FPGAs. In *Application Specific Processors, 2008. SASP 2008. Symposium on* (pp. 101-107). IEEE.

[38] Stone, J. E., Gohara, D., & Shi, G. (2010). OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, *12*(1-3), 66-73.

[39] Khronos OpenCL Working Group. (2008). The opencl specification. *version*, *1*(29), 8.

[40] Rogers, P. (2013). Heterogeneous system architecture overview. In *Hot Chips* (Vol. 25).

[41] Barr, M. (1999). *Programming embedded systems in C and C++*. " O'Reilly Media, Inc.".

[42] Kamal, R. (2008). *Embedded systems 2E*. Tata McGraw-Hill Education.

[43] Deschamps, J. P., Bioul, G. J., & Sutter, G. D. (2006). *Synthesis of arithmetic circuits: FPGA, ASIC and embedded systems*. John Wiley & Sons.

[44] Sriram, S., & Bhattacharyya, S. S. (2009). *Embedded multiprocessors: Scheduling and synchronization*. CRC press.

[45] Lee, I., Leung, J. Y., & Son, S. H. (Eds.). (2007). *Handbook of real-time and embedded systems*. CRC Press.

[46] Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., ... & Stenström, P. (2008). The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, *7*(3), 36.

[47] Bavier, A. C., Montz, A. B., & Peterson, L. L. (1998, June). Predicting MPEG execution times. In *ACM SIGMETRICS Performance Evaluation Review* (Vol. 26, No. 1, pp. 131-140). ACM.

[48] Yang, J., Ahmad, I., & Ghafoor, A. (1993, August). Estimation of execution times on heterogeneous supercomputer architectures. In *Parallel Processing, 1993. ICPP 1993. International Conference on* (Vol. 1, pp. 219-226). IEEE.

[49] Giusto, P., Martin, G., & Harcourt, E. (2001, March). Reliable estimation of execution time of embedded software. In *Proceedings of the conference on Design, automation and test in Europe* (pp. 580-589). IEEE Press.

[50] Li, T., & John, L. K. (2003). Run-time modeling and estimation of operating system power consumption. *ACM SIGMETRICS Performance Evaluation Review*, *31*(1), 160-171.

[51] Sykes, A. O. (1993). An introduction to regression analysis.

[52] Doss, R. G. (2011). Exploring Approximation Algorithms and Their Empirical Analysis for Selected NP-Complete Problems (Doctoral dissertation, Northcentral University).

[53] Doss, R. G. Last accessed 14/10/2015 http://www.codeproject.com/Articles/357645/Multiprocessor-Scheduling-Using-Partition-Approxim

[54] Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A k-means clustering algorithm. *Applied statistics*, 100-108.

[55] Hartigan, J. A. (1975). *Clustering algorithms*. John Wiley & Sons, Inc..