

**Multicore Architecture Optimization Using Novel Smart
Parallel Algorithms for Steganography and Image
Feature Extraction**

by



TSHOLOFELO MOGALE

(STUDENT NUMBER: 21887225)

**A thesis submitted in conformity with the requirements for the
the degree of Master of Science (MSc)**

Department of Computer Science

North-West University

Mafikeng, South Africa

**Supervisor: Prof O.O Ekabua
Co-Supervisor: Prof M.B Esiefarienrhe**

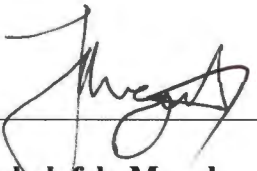
June, 2015



DECLARATION

I, TSHOLOFELO MOGALE hereby declare that this project report titled , “ *Multicore Architecture Optimization Using Novel Smart Parallel Algorithms for Steganography and Image Feature Extraction* “ is my own work carried out at North West University, Mafikeng Campus and has not been submitted in any form for the award of a degree to any other university or instituion of tertiary education or published earlier. All the material used as source of information has been acknowledged in the text.

Signature: _____


Tsholofelo Mogale

Date: _____

16/10/2015

APPROVAL:

Signature: _____



Date: _____

16/10/2015



SUPERVISOR: Prof. O.O. Ekabua

Department of Computer Science
North West University
Mafikeng Campus
South Africa

CO-SUPERVISOR: Prof. M.B Esiefarienrhe

Department of Computer Science
North West University
Mafikeng Campus
South Africa

DEDICATION

I dedicate this research dissertation to my class teacher:

Mr. T. Thipe,

And my late mother:

Ms. Phemelo Joyce Mogale,

Thank you for always believing in me and my education.

ACKNOWLEDGEMENTS

Before I say anything I would like to thank God almighty for everything he has done for me and for every opportunity he has given to me.

There is a fine line between saying something and doing something and it took 20 years for me to finally cross that line.

Firstly, I owe so much to my supervisor, Professor O. O. Ekabua. Thank you Prof for everything and may God bless you with many more years of success and prosperity.

Secondly, I would like to extend my appreciation to Professor M.B. Esiefarienrhe for taking every effort to see that this work is complete.

Thirdly, I would like to thank the staff members of Computer Science department for all the guidance and motivation through all the challenges I faced. Most notably I would like to thank Nosipho Dladlu, Thuso Moemi, Mmoloki Mangwala, Dr. Naison Gasela, Ifeoma Ohaeri and Frank Lugayizi.

To all my classmates in the Masters Class of 2015, thank you for being the brothers and sisters whom I relied upon in times of need.

To our family, Letlhogonolo Mogale, Onalenna Mogale, Refilwe Mogale, Kagiso Mogale, Tshepo Mogale, Omphile Mogale and Lebogang Mogale as always your support was the best thank you for believing in me.

I would like to give a special thank you to Boitumelo KaboEntle Molefe, Heavenly God knows that this work is complete because of the love and compassion she shared throughout. I will forever remain in her debt.

To my Best Friend Thato Makatong, thank you for everything. God is great.

Thank you all.

Abstract

Applications of Steganography and Image Feature Extraction are widely used and adopted by many organizations in the image processing industry. Many of these applications involve compute-intensive tasks which demand for full processor power. This research work intends to intervene on the performance problems that are experienced by algorithms in both steganography and Image Feature Extraction over Multicore Architectures systems. The main goal of this research is to provide Novel and Smart parallel algorithms in Steganography and Image Feature Extraction that are optimized to obtain enhanced performance in Multicore Architectures. The objective is to design, optimize and implement Novel Smart Parallel Algorithmic Models that will fully utilize Multicore Architectures. The results show high performance throughput as compared to ordinary algorithms which can only span about 433 samples when in execution. We have successfully developed algorithms which can span about 32,642 samples in execution. From these observations we conclude that Multicore Architectures Optimization is a necessity for the scaled performance of Steganography and Image Feature Extraction algorithms.

Table of Contents

DECLARATION	i
DEDICATION	ii
ACKNOWLEDGEMENTS	iii
Abstract.....	iv
List of Figures.....	viii
List of Tables	x
List of Acronyms	xi
Chapter 1:	
Introduction.....	1
1.1 Introduction and Background.....	1
1.2 Problem Statement	4
1.3 Rationale of the Study	5
1.4 Research Questions	5
1.5 Research Goal.....	6
1.6 Research Objectives	6
1.7 Literature Survey	6
1.8 Research Methodology	8
1.8.1 Literature Review	8
1.8.2 Model Design and Formulation	8
1.8.3 Metric Development	8
1.8.4 Model and Algorithm Implementation.....	8
1.8.5 Algorithm Analysis and Evaluation	8
1.9 Chapter Summary.....	8

Chapter 2:

Literature review	10
2.1 Chapter Overview.....	10
2.2 Preamble	10
2.3 Multicore Architectures	10
2.4 Multicore Architectures Specifications	15
2.5 Parallel Programming Models	18
2.6 Steganography with Parallelism	21
2.6.1 Steganography with Parallelism	22
2.6.2 Analytic Review of Steganographic Algorithms	24
2.7 Image Feature Extraction with Parallelism.....	25
2.8 Canny Edge Detector (CED)	26
2.8.1 Analytic review of CED algorithms	28
2.9 Conclusion	29

Chapter 3:

Model design, and algorithms implementation	30
3.1 Chapter Overview.....	30
3.2 Model Design and Approach	30
3.2.1 The Shell	31
3.2.2 The Kernel	31
3.3 Model and Algorithm implementation.....	35
3.3.1 Software Tools.....	35
3.3.2 Hardware setup.....	36
3.4 Conclusion	42

Chapter 4:

Model results and discussion	43
4.1 Chapter Overview.....	43
4.2 Image Feature Extraction Results.....	43
4.2.1 CPU Sampling And Total Usage	43
4.2.2 Total CPU Usage Per Core	45
4.2.3 Context Switch Per Core.....	46
4.3 Steganography Results	48
4.3.1 CPU Sampling And Total Usage	48
4.3.2 Total CPU Usage Per Core	49
4.3.3 CPU Utilization by Process.....	50
4.5 Critical Analysis.....	51
4.6 Conclusion	54

Chapter 5:

Summary, conclusions and future work	55
5.1 Chapter Overview.....	55
5.2 Summary	55
5.3 Conclusion	57
5.4 Future Work	57
Appendix A	58
Appendix B	62
References.....	66

List of Figures

Figure 1.1	Overview of NGN Architecture	02
Figure 2.1	Four Layer VNIX Design.....	12
Figure 2.2	Multicore Architecture CPU Chip Design	13
Figure 2.3	Moore's Law.....	14
Figure 2.4	Evidence of the Power Wall.....	15
Figure 2.5	Multicore Architecture Style and Chip Design	16
Figure 2.6	Core M family Microarchitecture	18
Figure 2.7	Flynn's Taxonomy	19
Figure 2.8	Amdahl's Law	20
Figure 2.9	Data Steganography Tree Structure	21
Figure 2.10	Steganography Overview	22
Figure 2.11	Simple LSB method on 32-bit Wav	24
Figure 2.12	Canny Edge Detection Algorithm	26
Figure 3.1	Node of the Proposed Model.....	31
Figure 3.2	The Kernel Layer	32
Figure 3.3	Dual Core Multicore Architecture.....	33
Figure 3.4	Quad Node	33
Figure 3.5	Octal Node	34
Figure 3.6	Medical Research Application	37
Figure 3.7	CED Applied on Medical Research Application.....	38
Figure 3.8	CED Applied on Other Medical Research Application.....	38

Figure 3.9	CED Applied on Computer Vision Research.....	39
Figure 3.10	CED with Computer Vision	39
Figure 3.11	CED Applied on Communication Research.....	40
Figure 3.12	LSB Applied on GIS Research.....	40
Figure 3.13	Screen Output for LSB Algorithm	41
Figure 3.14	The Generated Stegoimage	41
Figure 4.1	Suboptimal CPU Usage over Wall Clock Time	44
Figure 4.2	Optimal CPU Usage over Wall Clock Time	44
Figure 4.3	Suboptimal CPU Usage Per Core.....	45
Figure 4.4	Optimal CPU Usage per Core	46
Figure 4.5	Suboptimal Context Switch per Core.....	46
Figure 4.6	Optimal Context Switch per Core	47
Figure 4.7	Suboptimal CPU Usage over Wall Clock Time	48
Figure 4.8	Optimal CPU Usage over Wall Clock Time	48
Figure 4.9	Suboptimal CPU Usage per Core.....	49
Figure 4.10	Optimal CPU Usage per Core.....	49
Figure 4.11	Suboptimal CPU Usage by Process	50
Figure 4.12	Optimal CPU Usage by Process	50
Figure 4.13	Gustafson's Law.....	51
Figure 4.14	IFE Profiling per Thread.....	52
Figure 4.15	Steganography Profiling per Thread	53

List of Tables

Table 2.1	Multicore Architecture CPUs by Manufacturer.....	17
Table 2.2	Parallel Programming Models	18
Table 2.3	Simple LSB method on Colour Image Pixel.....	23
Table 4.1	Metric Summary Comparison for IFE and Steganography.....	54

List of Acronyms and Abbreviations

BMP	Bitmap
CBIR	Content Based Image Retrieval
FPGA	Field Programmable Gate Array
IFE	Image Feature Extraction
ILP	Instruction Level Parallelism
ISA	Instruction Set Architecture
JPEG	Joint Photographic Experts Group
LSB	Least Significant Bit
MIMD	Multiple Instruction Multiple Data
MSE	Mean-Squared Error
NUMA	Non Uniform Memory Access
NoC	Network on Chip
PNG	Portable Network Graphics
PSNR	Peak Signal to Noise Ratio

PPM	Parallel Programming models
SIMD	Single Instruction Multiple Data
SSE	Streaming Single Instruction Multiple Data Extension
TLP	Thread level Parallelism
TLTP	Thread level Task Parallelism
VM	Virtual Machine
VNIX	Virtual Machine

Chapter 1

Introduction

1.1 Introduction and Background

With significant changes currently present in the technology of microprocessor architectures, it is evident that the salient way to promote efficiency and improve performance is no longer by increase in clock speed [1] but by the use of Multicore architectures. Multicore architectures as seen in Figure 1.1, enable Multicore processing which allows the use of more than one processor (core) with shared cache on a single chip to carry out tasks efficiently. However, this major change in technology has ended the concept of a free ride that programmers have enjoyed for decades whereby their applications continue to improve in performance automatically whenever new CPUs came out of the box with increased clock frequency [1]. Programmers and researchers now have to revisit their initial algorithm designs and have them optimized to harness the power of Multicore architectures [2]. The approach now is to use a branch of concurrent computing known as parallel computing. In particular programmers and researchers have to use parallel programming on their algorithms because Multicore architectures are parallel by nature [3]. Steganography is known as the art of hiding information in information. Steganography hides the fact that information or communication exists. Steganography is not a new study; it has been studied and practiced for decades [4].

Steganography forms a branch of information security which deals with information hiding primarily concerned with concealing the existence of information. Steganography differs from cryptography in that cryptography provides a way to protect data from being compromised, while Steganography eliminates awareness of the fact that the data exists, for security purposes [4]. The former can employ the latter in the case when data is in danger of being compromised and more security is needed to protect the data. There exist different forms of Steganography. There is text-steganography, image-steganography, audio-steganography and video-steganography. The files used for Steganography are called covers and these are used to protect data from being easily detected. The most widely used covers are image covers because of their consistency and their frequency over the internet [4].

The other reason why image covers are used is because of the data structure and compression standards that digital images are often decoded on [5]. Compression Standards such as JPEG, PNG, BMP often provide a robust canvas for concealing embedded data [4]. Many algorithms implementing different techniques of Steganography have been proposed by researchers [5, 6]. Most of these algorithms are serially implemented and contain compute intensive tasks, as evident in [7]. This is not a problem if the data to be protected is not large in terms of quantity. However if the data to be processed is in large quantities this can become tedious and time consuming work. Further, since the algorithms are implemented serially, this leads to poor performance and scaling. Implementing such serial algorithms in Multicore architectures which are commonly found on almost every personal computer of today will result in poor performance [1].

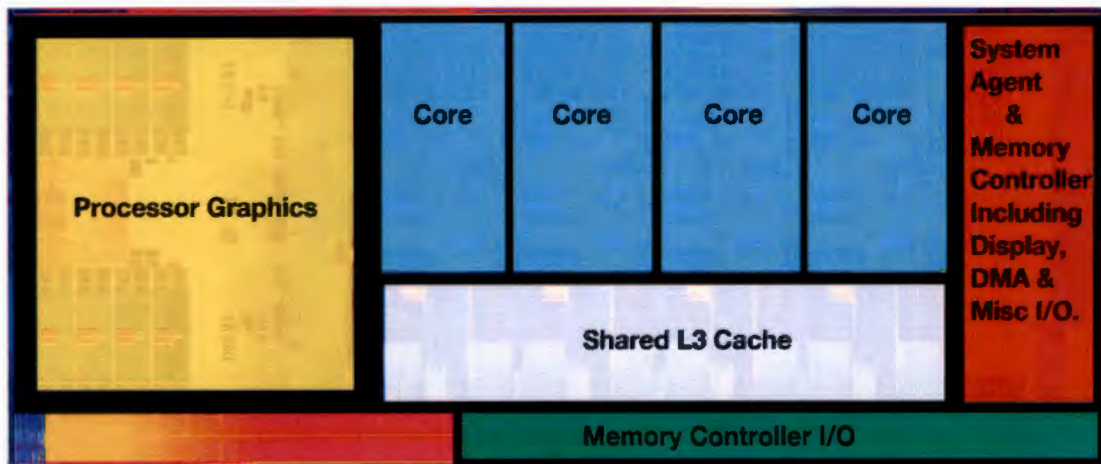


Figure 1.1: Multicore Architecture with Four Cores [1]

Image Feature Extraction (IFE) is commonly used in many image processing applications. It is concerned with the extraction of features that are found on digital images which include colour, texture, shape and form. There are many applications where IFE is applied. Common applications include Content Based Information Retrieval (CBIR), Image Steganography applications and computer vision applications. All these applications rely on feature detection because the primary goal of IFE is met through feature detection. CBIR retrieves images by visible information such as colour, texture and shape for some special types of images [8]. Human beings make use of feature detection on a daily basis to make sense of objects they come across.

This is made possible through detecting feature components of an object such as colour, form and motion. The cones found in the retinas of our eyes give us the ability to make sense of colours. The Parvo pathway in our brain helps us with figuring out the form or the shape of the object, while the Magno pathway allows us to encode the motion of an object. The interesting fact is that as humans we have the ability to detect all three features at the same time when we look at any object. We do not first process the colour, the form then the motion. We process all of them simultaneously, and this is known as parallel processing. It becomes possible because our brains are parallel by nature. The same can never be said for computer systems when dealing with feature detection.

The amount of work to be done just for detecting the colour and the form in an image is very huge and complex. As illustrated by the authors in [8] IFE is a time consuming process and this is even worse when the images to be processed are in large quantities. Consequently this problem is often encountered by image processing applications on the Internet because of the high frequency of image data on the Internet. Steganalysis is primarily concerned with attacking Steganography. Researchers in [9] have identified that the calculation of IFE algorithms constantly increases, and this contributes the most time consuming step in image steganography detection. Researchers in [8] discovered that most IFE methods do not care much about performance and do not take note of the utilization of the highly developed microprocessor architectures. Almost all image processing applications are implemented serially and this leads to poor results in terms of performance, even on the highly developed microprocessor architectures of modern day computer systems.

This work intends to contribute to solving the performance problems that are often encountered by algorithms in both steganography and Image Feature Extraction. The main strategy of this work relies on utilizing parallel architectures with parallel programming tools to tackle the performance related problems evident in both steganography and Image Feature Extraction algorithms. The challenges will be addressed by first optimizing the algorithm designs to take full advantage of Multicore architectures. Then various subjective and objective experiments will be performed to validate the algorithm designs and to assess the issues pertaining to degrees of performance. In doing so, this work intends to contribute novel smart parallel algorithm designs that are optimized to leverage Multicore architecture

platforms in order to propel performance in applications of Steganography and Image Feature Extraction.

1.2 Problem Statement

Applications of Steganography and Image Feature Extraction are widely used and adopted by many organizations in the image processing industry [5]. Many of these applications apply different methods of which many involve several compute intensive tasks as visible in [5, 7, 6, 10, 11, 12]. These methods are valid for small quantities of image data but pose a problem when the image data quantity is large. Over the years Multicore architectures have emerged to be ubiquitous in the computer industry. Still most of the existing algorithms and methods do not seem to care about performance or utilizing Multicore architectures to increase performance [8, 13]. For example the internet carries billions of image data and most of these methods would not be appropriate for handling such challenging tasks in real time [3]. Hence, optimizing Steganography and Image Feature Extraction algorithms to leverage Multicore architectures is mandatory but still remains a critical problem. Moreover, there is no defined list of guidelines that can be followed for optimizing every application or algorithm and this makes deploying applications for Multicore architectures a very challenging task.

The questions that arise from these concerns are:

- a) How can we leverage Multicore architectures to improve the performance of algorithms in image processing applications?
- b) Can Steganography and Image Feature Extraction algorithms be optimized to fully utilize Multicore architectures to increase performance?
- c) What level of effectiveness can be achieved by utilizing Multicore architectures for algorithms and applications in Steganography and Image Feature Extraction?

1.3 Rationale of the Study

Since Multicore architectures revolutionized the microprocessor industry, in most of the proposed algorithms and applications, only a few in the image processing industry have taken notice of the utilization of these architectures to increase performance. In most of the algorithms proposed by researchers, performance still remains a major problem. Although some research has been done on implementing different strategic methods for algorithms in Steganography and Image Feature Extraction [4, 5, 7], the main focus for researchers has been to come up with different strategic methods for these algorithms, rather than focusing on the effectiveness of the algorithms in terms of performance.

Various methods for optimizing algorithms in steganography and Image Feature Extraction have been suggested by several researchers. Ming Qi and his team [8] introduced some optimized methods of Image Feature Extraction, which include both Thread Level Parallelism (TLP) and Simple Instruction Multiple Data (SIMD) Instruction Level Parallelism (ILP). They analyzed their methods before and after the optimizations and their experiments indicated that a good speedup ratio is gained by TLP and SIMD ILP optimizations. Following this work, Chenjun and Shangping [9] proposed a Stego feature extraction method that used Thread-Level Task Parallelism (TLTP) and the results of their work showed good improvements in terms of performance speedup ratios on the dual-core and quad-core systems.

The goal of a researcher in modern computing environments should not be to take advantage of Multicore architectures with two or four cores, but instead it must be to design and implement scalable algorithms that will take advantage of any amount of parallel hardware available.

1.4 Research Questions

The questions that need to be answered in order to meet the goal of this research are:

- a) How can we utilize Multicore Architectures to improve performance?
- b) How can we develop efficient smart parallel algorithms that are optimized to fully utilize Multicore Architectures to address performance problems?

c) How can we implement these algorithms in Steganography and Image Feature Extraction to achieve good performance and scalability?

1.5 Research Goal

The main goal of this research is to provide Novel and Smart parallel algorithms in Steganography and Image Feature Extraction that are optimized to obtain enhanced performance in Multicore Architectures.

1.6 Research Objectives

To achieve the main goal of this research, the following objectives shall be employed:

- a) Design Novel Smart Parallel Algorithmic Models that will fully utilize Multicore Architectures.
- b) Optimize the designed Parallel Algorithms to scale for applications of Steganography and Image Feature Extraction applications on Multicore Architecture environments.
- c) Implement the designed and optimized Parallel Algorithms on Multicore architecture environment to improve overall performance of Steganography and Image Feature Extraction.

1.7 Literature Survey

Almost every personal computer today has parallel features or supports parallelism. All modern computer designs have parallel hardware which either features Multithreaded cores, Multicore processors, or Graphical Processing Units (GPUs). There has been a substantial amount of research conducted on optimization of image processing algorithms for parallel hardware in general. Kaur [7] implemented image processing algorithms on the parallel platform using Matlab. In his work he shows that the major challenge of parallel processing is not only to aim for high performance, but also to give the solution in less time with better utilization of resources. The results of his work show that by leveraging Multicore platforms can fairly speed up the processing of images. Perhaps the most prominent work is the one conducted by Agarwal and other scholars in [2]. They proposed

a set of optimizations for bilateral filtering kernel and they introduced a pair-symmetric algorithm which has the theoretical potential to reduce processing time by half by exploiting the availability of special purpose registers in Multicore machines.

Optimization of algorithms for Multicore Architectures cannot only be static, but must also be dynamic. Dynamic optimization works during program execution and this can lead to too much complexity. Several dynamic optimizers have been proposed by researchers in literature, however the most notable is one presented by Lin, Liu and Wu in [14]. In this work they argue that since Multicore architectures share cache among cores as evident in Figure 1.1, it raises a lot of competition among threads for cache. They conclude that this competition for shared cache among threads may affect their behaviour and degrade the overall performance. Hence they recommend that this competition can only be handled by a dynamic optimizer during runtime. However this is not necessary for node and vector parallelism. This competition for cache by threads can also be remedied by the use of a strategic work-stealing and load-balancing scheduler depicted in [1, 15].

Ming Qi, Sun and Chen in [9] successfully tried a methodology that integrated TLP and SIMD ILP optimization, to utilize Multicore Architectural feature and SIMD feature of the modern CPU. It would be interesting to find out how the same could be achieved for Multiple Instruction Multiple Data (MIMD) machines. Ling and Zhong [9] focused on developing a Stego feature extraction method that uses TLTP, which first constructs a lock-free task queue for task parallelism, then reduces thread synchronization overhead and finally solves the false sharing issue by setting up a thread affinity scheduling algorithm to improve performance [9]. Taking all this remarkable work into consideration, the ultimate goal remains to provide Novel and Smart parallel algorithms in Steganography and Image Feature Extraction that are optimized to obtain enhanced performance in Multicore Architectures.

1.8 Research Methodology

The following methodology shall be followed in order to successfully conduct this research:

1.8.1 Literature Review

A Formal Literature Survey on previous works including the strategic methods and techniques related to the research will be undertaken.

1.8.2 Model Design and Formulation

Based on the surveyed literature, the design of Novel Smart Parallel Algorithmic Models that are optimal for Multicore Architectures will be developed.

1.8.3 Metric Development

A performance metric will be adopted to evaluate and benchmark the performance of the model and algorithm.

1.8.4 Model and Algorithm Implementation

Based on the designs, this research will implement the Novel Smart Parallel Algorithms for Steganography and Image Feature Extraction on Multicore Machines as proof of concept.

1.8.5 Algorithm Analysis and Evaluation

An analysis and evaluation of the designed Novel Smart Parallel Algorithms will be conducted for performance and scaling on Multicore Architecture Machines.

1.9 Chapter Summary

This chapter presented an overview and general insight of the research work. In this chapter a general understanding of the research work was outlined together with a research plan which included a clear presentation of the problem statement, research objective, research goal, research methodology and milestones.

The remainder of the research work is organized as follows:

Chapter 2 - Literature Review:

This chapter will provide detailed review of the work done currently and previously by researchers as basis for the current work.

Chapter 3 - Model, Algorithm and Metric Development:

This chapter shall present the designed model, algorithms and the metric developed. A detailed report of the components and variables will be given. The chapter will also demonstrate how the model, algorithms and metric work together to achieve the main goal of this research.

Chapter 4 - Implementation:

This chapter will give the implementation details of the model, algorithm and metric. In conclusion it will give a detailed report of the implementation and analysis of results.

Chapter 5 - Summary, Conclusions and Future Work:

This will give the summary of the work and also give a conclusion based on the results obtained. Finally potential future work will be outlined.

Chapter 2

Literature review

2.1 Chapter Overview

This chapter first gives a detailed review of Multicore architectures with respect to the surveyed literature published over the years by other scholars. Also this chapter reviews and discusses parallel algorithms and the level of parallelism that can be achieved by utilizing Multicore Architectures. In conclusion a summary of the chapter and a brief outline of the next chapter is given.

2.2 Preamble

In order to meet the objectives of this research a theoretical understanding of Multicore architectures together with the principles of parallel programming must be achieved. These principles shall enable us to implement Novel Smart Parallel Algorithms in Steganography and Image Feature Extraction in Multicore architectures. We first start by reviewing and discussing Multicore architectures.

2.3 Multicore Architectures

In recent years Multicore architectures have evolved eminently from the traditional Monolithic single-core processors. This evolution of Multicore architectures was sparked by the impacts of limits encountered by microprocessor developers over the years. Several of these limits could be ignored, however the most notable ones, that architects could not ignore, are the ones discussed by authors in [1] known as the “three walls”.

The first of these walls to be encountered or realized was the Power wall, as a result of unacceptable growth in power usage with clock rate, and also the realization was that above around 130W air cooling is not sufficient [16, 1]. Second was the ILP wall, due to the limits at which one can achieve low-level parallelism. The last wall is the Memory wall which resulted because processor speeds were highly discrepant to memory speeds. The most significant of all of the three walls is the power wall. This had a very large impact in the microprocessor industry and was perhaps the most compelling factor for architects to shift

processor designs to Multicore architectures, successfully led to the establishment of the Multicore era.

There has been a substantial amount of literature published on Multicore architectures and the research is on-going [2, 8, 9, 14, 17].

Saleem et al [13] optimized a parallel quick sort algorithm for Multicore architectures using Intel Cilk plus. Sorting algorithms are known to be quite complex and can take impractical times when applying them on large data sets. However in [13] they were able to sort one million elements in 0.218 seconds and this span decreased as the number of cores (workers) increased. Hence, their work reports that vast amounts of speed up ratios can be achieved by increasing core count. In future they intend to optimize a parallel program for merge sort.

Research on Multicore architectures is rapidly increasing. Chhugani et al [18] have taken the lead and proposed an efficient implementation of merge sort algorithm on Multicore Architectures with SIMD technology. In their work they showed that their SIMD implementation which featured 128-bit Streaming SIMD Extension (SSE) was 3.3 times faster than the scalar version. This implementation was able to sort 64 million floating point numbers in less than half a second on a simple commodity 4-core processor. The result served as a testimony that, even though there is a taxonomy of sorting algorithms, different architectures require custom implementation crafted suitably for them to achieve efficient sorting. Their research also showed that modern shared-memory (Shared cache) Multicore architectures with SIMD instructions can perform high performance sorting, which used to be possible on message passing machines only and required a message passing interface for programming [3, 18].

While below the same horizon of utilizing SIMD instruction in Multicore Architectures for elevated performance, Klemm et al [19] extended OpenMP library with vector constructs for optimal performance on modern Multicore architectures with SIMD. In this work they propose a new OpenMP directive that will help programmers to guide the vectorization process, enabling portable exploitation of SIMD levels in Multicore Architectures. This extension constructs a bridge and closes the gap of overhead for programmers because it allows for OpenMP to extend Thread Level Parallelism (TLP) to Instruction Level Parallelism (ILP) [19]. By evaluating with a set of benchmarks they were able to show how

the new directive can improve performance over the traditional auto-vectorizer of compilers. In future they aim to improve the vector code generation and also test portability for other platforms. While Klemm and team in [19] put much effort on a directive that focused on instructing a compiler on which loops should be vectorized, Kozhukhov et al [20] extended the focus not only to loops, but also to functions. Since loops can also be encountered inside functions, their main idea is to vectorize the entire function instead of the loops inside a function.

Optimization for Multicore architectures can also be used to improve performances of algorithms and software that already exist. As already outlined by scholars in [1, 3, 9, 15, 21], the main cause for improvement relies on exploitation of parallelism in Multicore architectures, since they are parallel by nature. Most of the currently existing software is limited in performance because of the initial design patterns adopted during the developmental stage. This subjects them to low levels of parallelism and performance rates on Multicore architecture technology. As an aid to the aforementioned problems, Xuanhua et al [22] proposed a Virtual Machine (VM)-based Web system on Multicore clusters that is scheduled by a Linux Virtual Server. Their VM-based Web system is oriented by VNIX, which is a set of VM management toolkit developed by his team.

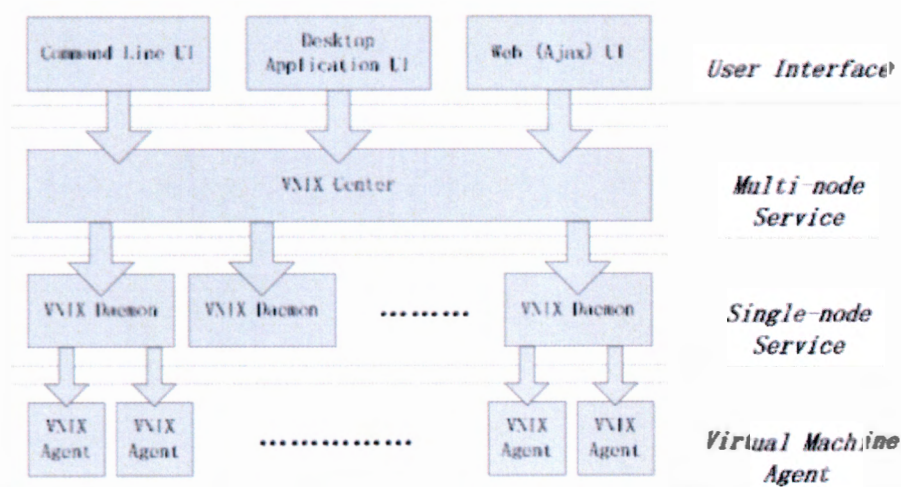


Figure 2.1: Four Layer VNIX Design [23]

VNIX [23] was developed mainly for facilitating the management of VMs on clusters, while at the same time it aimed at improving the usage of Multicore Architecture CPUs on nodes. The results of the experiments conducted in [22] show that VM-based Web systems perform about three times faster than classical web systems deployed on Multicore Clusters. Web systems are communication applications and this means that Multicore Architectures can guarantee performance improvements for different types of applications in different domains. While in the spectrum of communication applications, Ruijin et al [24] proposed a novel Multicore processor with SIMD, Instruction Set Architecture (ISA) and an extended register file for communication applications as seen in Figure 2.2. This CPU will be suitable for communication applications such as the one proposed by Xuanhua and his team in [22] since it is crafted ideally for such applications.

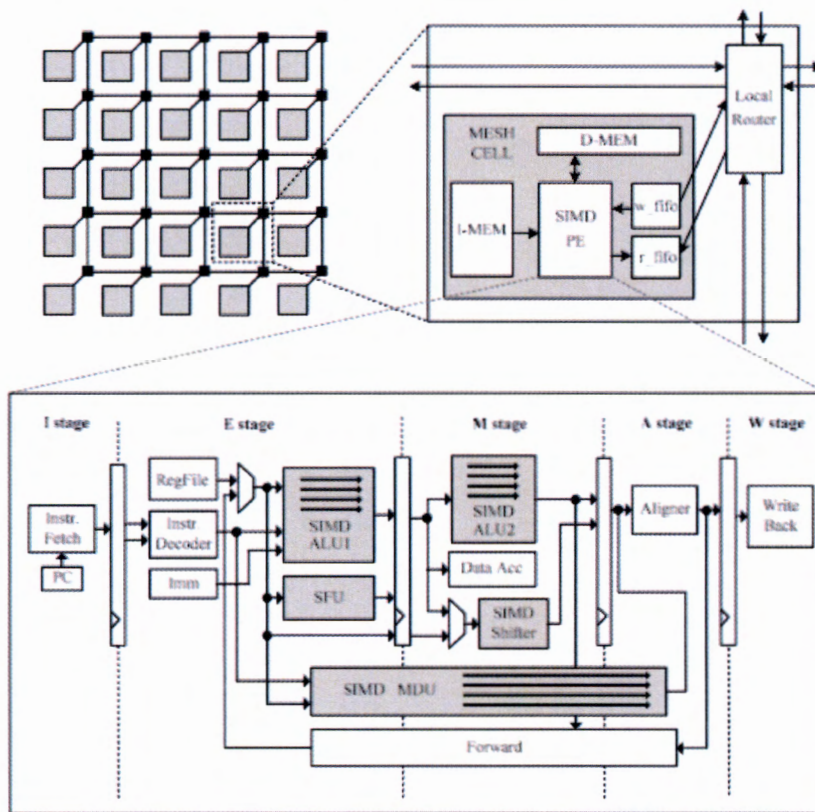


Figure 2.2: Multicore Architecture CPU chip design [24]

Moore's Law as mentioned by authors in [16, 1, 3, 25] dictates. As shown in Figure 2.3 Moore's law bestows performance for any chip that increments its overall transistor count. To obtain high computing power Ruijin and his team implemented SIMD ISA and

increased the register file count from 32 to 64. Furthermore, they adopted a 5x5 homogeneous 2-D mesh NoC (Network-on-Chip) topology to boost parallelism. To test the processor they used a Reed-Solomon (RS) decoding algorithm with strength of 255, 2398 for evaluating the proposed Multicore CPU performance. The obtained results showed that the CPU achieved 2.175 Gbps of throughput in worst case scenario in 8 error version of RS algorithm. Hasita et al [26] proposed an FPGA-based Implementation of Heterogeneous Multicore platform with custom accelerators for power-efficient computing. Since an optimal architecture would differ from one application to the other, it is imperative to explore suitable architectures for different types of applications. Hence, the proposed platform in [26] allows one to select the most suitable accelerator according to application requirements. Their results show power-efficiency that is 15 times more efficient than that of traditional GPU.

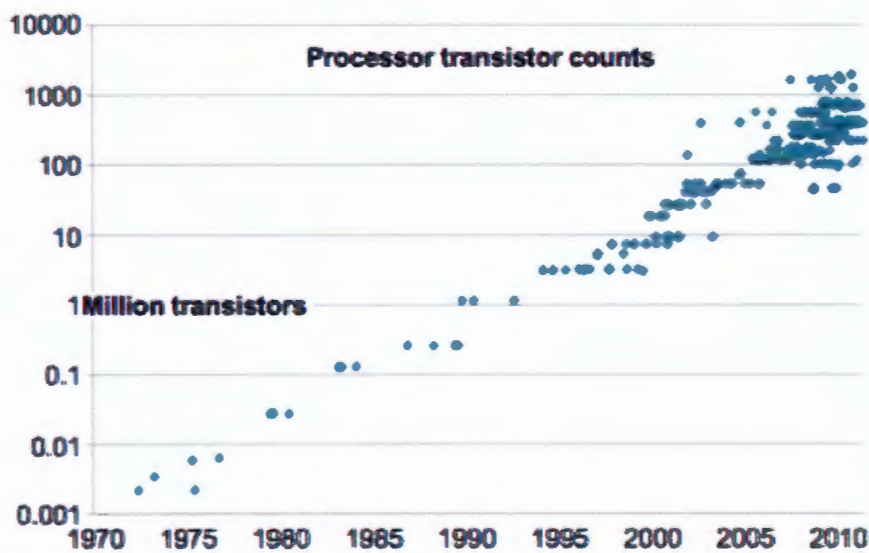


Figure 2.3: Moore's Law [1]

A major disadvantage of Multicore Architectures is that, as much as they can be used to alleviate performance issues in applications, they can also be the major downfall for the performance of applications if not well utilized [16]. In order for one to fully exploit their parallelism one must know what their architectures are composed of and, since they vary in design, one also needs to know which programming model is suitable for use as an exploitation tool. Overlooking and not fully understanding these compositions can be very pejorative to the performance of applications on a parallel platform [16, 1, 3].

In the next section, a discussion of the Multicore architecture's architectural composition and the different technologies that complement them is presented. Following that is a discussion of the programming models that are suitable for exploiting and optimizing algorithms to take advantage of their parallel capabilities.

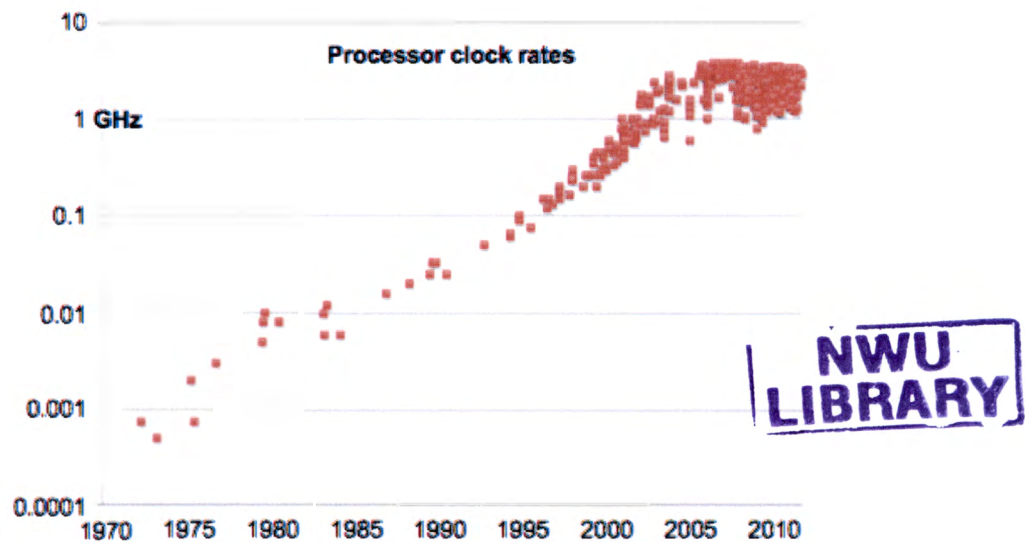


Figure 2.4: Evidence of the Power Wall [1]

2.4 Multicore Architectures Specifications

Perhaps the most compelling factor that motivated microprocessor architects to revise chip designs and opt for multicore was a phenomenon that is known as the power wall [1]. As seen in Figure 2.4 the power wall which display peaks of clock speeds. Researchers discovered that air cooling is not sufficient at around 130w [1]. This retired dreams of having microprocessor chips clocking at speeds of about 4 to 5GHz for the computing industry and the scientific community at large. Necessity is the mother of invention, hence architects saw it was inevitable to opt for multicore technology [27].

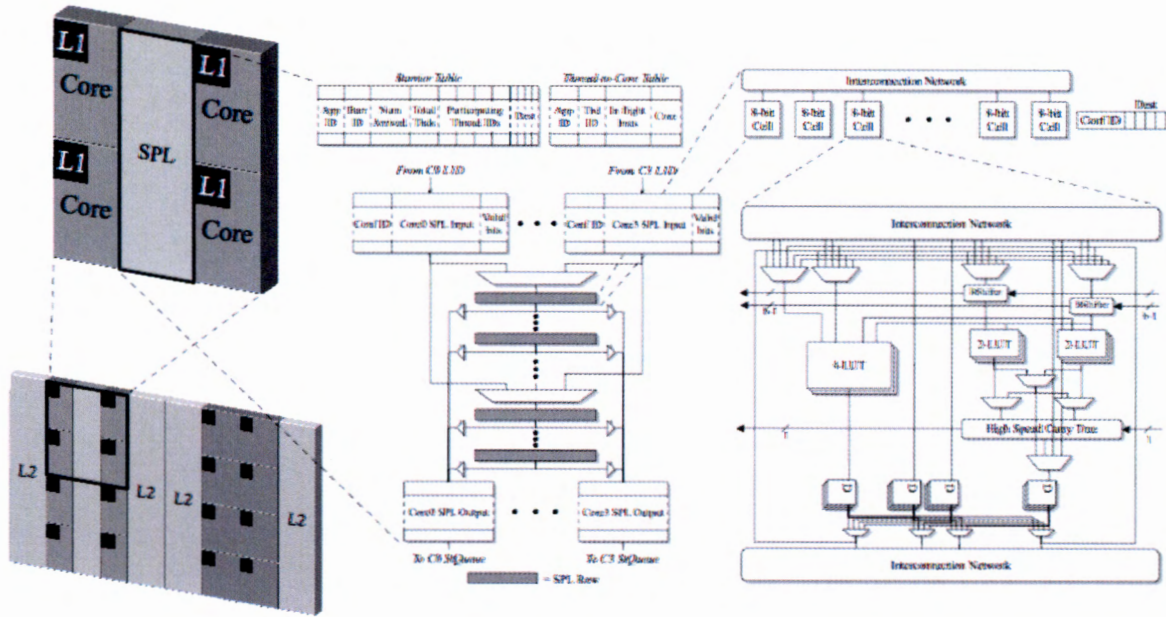


Figure 2.5: Multicore Architecture Style and Chip Design [28]

Multicore architectures, unlike the traditional monolithic uniprocessors, are simple in terms of architectural style but complex in terms of the overall design. As seen in Figure 2.5, the chip's design is quite sophisticated compared to the coined style presented in Figure 1.1. Because the chip now has multiple cores, designers have to account for how the cores communicate and also how they sufficiently share cache among themselves [28], hence the inclusion of an extra set of technologies which makes their designs quite sophisticated. Multicore architectures can be specified accordingly with the use of Table 2.1. Table 2.1 shows a wide variety of different CPUs with Multicore Architectures different manufacturers in the computing industry. They all adopt different styles, hence some chips are unique but they all have similar architectures.

Table 2.1: Multicore Architecture CPUs by Manufacturer

CPU	Frequency	Number of cores	Manufacturer
Athlon II X2 225	3.1 GHz	2	AMD
Phenom II X4 965	3.6 GHz	6	AMD
Scorpion	2.1 DMIPS/MHz	2	Qualcomm
Cortex-A15	2.5 GHz	2	ARM
Atom-Z520	2.13 GHz	2	Intel
Pentium M	2.26 GHz	2	Intel
Core i3-Sandy Bridge	3.4 GHz	4	Intel
Core i5-Haswell	3.4 GHz	4	Intel
Core i7-Guiltown	3.6 GHz	6	Intel

Due to its narrow scope, this research will survey only CPU Chips manufactured by Intel Corporation. In particular, the focus shall only be on the prime high-end CPUs of the core series, namely Core-i3, Core-i5, and Core-i7. Intel's current resolution of the semiconductor device fabrication node technology is at 14nm, and projections indicate they will it will shift to 7nm by 2020. The current migration to 14nm resolution will open doors to a wide variety of technology such as laptops and computers with fan-less designs. A set of cores have been proposed and these cores will be of 5th generation all featuring fan-less systems, multicores and advanced optimized graphics engines. The new series is named Core-M featuring a Multicore Architecture code named Broadwell. In Figure 2.6 the sample microarchitecture of a Core M family of CPUs is shown.

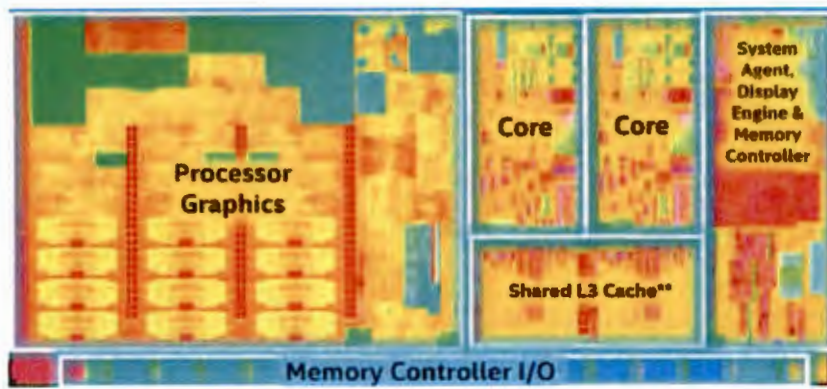


Figure 2.6: Core M family Microarchitecture

The Core M family uses TDP Reduction which enables fan-less designs. They feature 14nm second generation Tri-Gate Transistors. These features will enable smartphones, and laptops incorporate thinner designs since there wouldn't be a need for cooling systems. The establishment of these kinds of chips will give rise to new technology. In the next section we review Parallel Programming models suitable for Multicore Architectures.

2.5 Parallel Programming Models

A variety of Parallel Programming Models (PPMs) suitable for custom multicore architectures exist [29]. The models simply mirror an abstraction of the computer system architecture, therefore this makes computation to be highly optimized for any computing devices such as GPU, APU or CPU. Because of this mirrored abstraction, Parallel programming models are not tied to any object model and as such can have a wide area of use and application. PPMs can be applied for different platforms such as Web and Heterogeneous systems. Web service systems such as SOA, REST, and Ice utilize and adopt PPMs so that they may not be tied to any object models. There is a large variety of PPMs and, in Table 2.2 we describe the major ones.

Table 2.2: Parallel Programming Models

Parallel Programming Model	Programming Model	System Architecture	Synchronization	Communication Model
Posix Threads	Threads	Shared Memory	Explicit	Shared Address
OpenMP	Shared Memory	Shared Memory	Implicit	Shared Address
MPI	Message Passing	Distributed Memory	Implicit or Explicit	Message Passing or Shared Address

Because parallel systems are polythetic [1, 3], a vast number of PPMs are available to cater for them and complement Parallel Programming Languages (PPLs), even though PPLs are

polysynthetic, they exist to help optimize computation for these parallel systems [29, 1, 30]. These languages utilize algorithmic skeletons and parallel patterns to simplify computation. McCool et. al in [1] outline in detail the implementation and efficient use of parallel patterns for parallel computation, while Gonzalez et al in [31] give a broad overview of the prevalent algorithmic skeletons suitable for multicore architectures.

Parallel Machines have been in existence for a long period of time. In 1966, it was Flynn in 1966 [3, 32] proposed a taxonomy for classifying parallel computers. He surveyed several parallel machines in existence at the time and concluded that they can be divided into four main classes as we see in Figure 2.7. This classification is basically concerned with the data stream (MISD, MIMD) and instruction stream (SISD, SIMD). MIMD machines are highly parallel machines [13], although not entirely suitable for all work [32]. Modern Multicore Architectures abstract and inherit some of the MIMD properties and incorporate SIMD technology for increased parallelism [27, 18, 20].



		Instruction stream	
		Single	Multiple
Data stream	Single	SISD	MISD
	Multiple	SIMD	MIMD

Figure 2.7: Flynn's Taxonomy [2]

Not all work can be parallelized, in other terms not all computation work for processors (workers) can be performed in parallel. No matter how many processors a parallel machine may house, there will always be some serial elision [1, 3]. Amdahl became pessimistic about serial elision [33, 1]. He pioneered the field of parallel algorithms and coined a law which became infamously known as the fundamental principle of Parallelism [1].

$$\text{Speedup} = 1/(1 - f) + \left(\frac{f}{n}\right) \quad (2.1)$$

The law simply states that **speedup** is determined by **F** (parallelizable portion of work), and that as the number of **N** (processors) approaches infinity, the serial work (1+f) dominates computation hence decreasing speedup rates. This can be seen visually in figure 2.8. The key approach in this research pays close attention to Amdahl's Law and takes it into account. Scholars in [33] spearheaded a study to confirm if Amdahl's Law held for multicore architectures and the study confirmed that Amdahl's Law still holds. From the study a realization surfaced that symmetric and asymmetric dynamic multicore architecture designs held high in performance in terms of speedup ratios. The next sections present Steganography algorithms in general, as well as Image Feature Extraction algorithms and how they can be parallelized to be accustomed for Multicore Architectures.

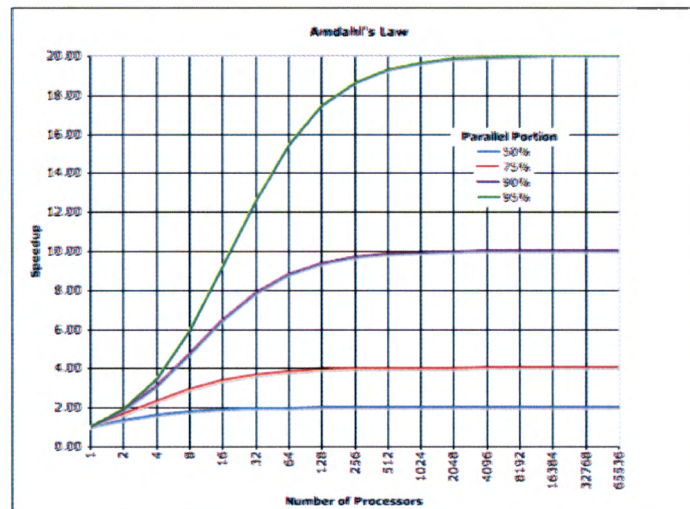


Figure 2.8: Amdahl's Law [1]

2.6 Steganography with Parallelism

Steganography, defined simply in a nutshell, is the science of hiding information in information [34]. The main golden rule about steganography is that data must remain concealed and must never be discovered or unconcealed or else that destroys steganography security. The science mainly concerned with the destruction of steganography security is called steganalysis [5].

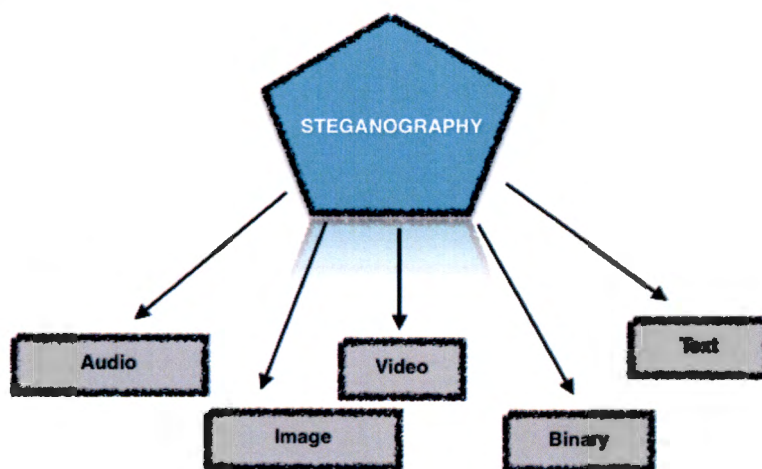


Figure 2.9: Data Steganography Tree Structure

Steganography hides the fact that information or communication exists [35]. Steganography differs from cryptography in that cryptography provides a way to protect data from being compromised, while steganography conceal the fact that the data exists for security purposes. The former can employ the latter in the case when the data is discovered and more security is needed to protect the data. Figure 2.9 shows that different forms of steganography exist, namely these are text-steganography, image steganography, audio steganography, binary steganography and video steganography. In addition, there is a variety of methods and algorithms for steganography, such as Least Significant Bit (LSB), Watermarking, Compression, Spread spectrum, etc [4]. Because steganography on its own is a main area of Information Security (Figure 2.10), the scope of this research is narrowed to Least Significant Bits steganography algorithms. This research aims to implement these algorithms in parallel on Multicore Architectures. The files used for steganography are called covers and these are used to protect data from being discovered [36]. The most widely used and easy to implement is image steganography, because everything has an image form, unlike sound and text. The other reason why images are used is because of the consistency and their frequency over the Internet. The motivating reason is because of the

compression standards that they are often found on and these include JPEG, BMP, PNG, etc. Secret messages can be hidden inside images and be sent over the Internet with no interference at all. The message can be encrypted in case the communication is tampered with or even eavesdropped. Solid steganography relies on good algorithms [5].

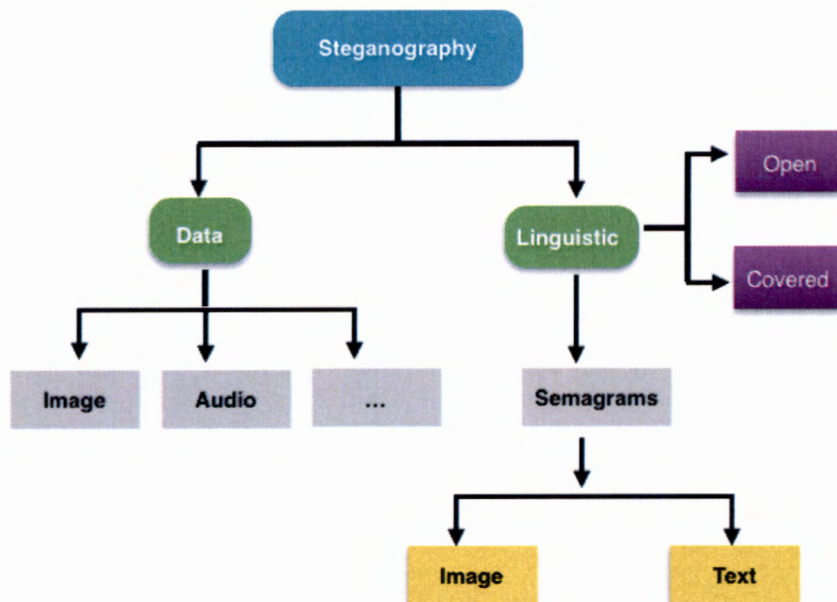


Figure 2.10: Steganography Overview

2.6.1 Least Significant Bit Steganography

The Least Significant Bit insertion method provides a simple way to embed messages in different covers of desire. LSB method involves manipulating bits of a cover and changing some to the bits of the cover file to accommodate the bits that contribute the overall message [6]. However LSB method can be very complex and lead to undesired results depending on the nature of the cover and the approach undertaken when implementing it. This method is quite common in fragile Steganography, and different approaches exist for implementing LSB [37], and this is normally dictated by the cover chosen as the message carrier. Billions of digital images of various types are found on the Internet. These digital images are either in black and white, gray scale or in full colour. Each image has a structure that is composed of picture elements (pixels). A black and white image pixel has a single bit that can either be zero or one. For grey scale images each pixel is defined in 8 bits of 0 (black pixels) to 255 (white pixels) and for Full colour each pixel has 24 bits each of 8 bits emphasizing RGB. Table 2.3 demonstrates simple LSB method for colour images.

Table 2.3: Simple LSB method on Colour Image Pixel

Colour Pixel element	8-bit binary Representation before insertion	Decimal Representation	LSB before Insertion	Colour of pixel before change	Message to Be Embedded	8-bit binary Representation after insertion	LSB after Insertion	Colour of pixel before change
R	11001100	204	0	Purple	"6"	11001101	1	Purple
G	10011001	153	1			10011001	1	
B	11111111	255	1			11111110	0	

The message "6" is embedded on the purple pixel of a digital image and the result after embedding is still a purple pixel with no visible change. Colour images are common on the Internet and are available in different formats. Different kinds of image formats vary in message embedding capacity [35, 5]. LSB when implemented with the right types of image formats can yield high message embedding capacity, and is also less prone to Steganalysis attacks.

LSB method can also be applied on audio signals [38] as seen in Figure 2.11. The message "H" is embedded on the 32 bit Wav in the least significant bits, mostly constituting audio noise at non audible frequencies. The message consists of a few bits that will alter the LSB bits of the digitized audio cover after embedding. In this research we focus entirely on optimizing LSB steganography on colour images for multicore architectures.

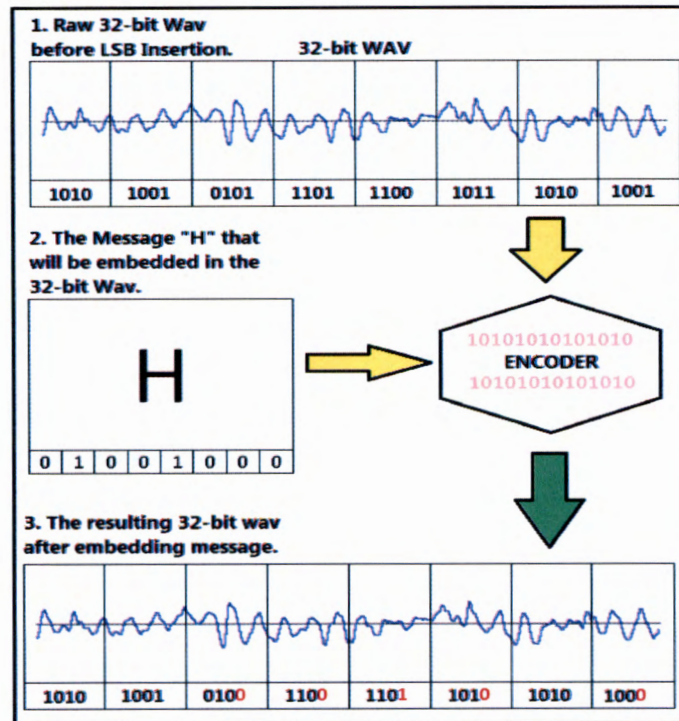


Figure 2.11: Simple LSB method on 32-bit Wav

2.6.2 Analytic Review of Steganographic Algorithms

So far the outputs of research published by scholars and researchers has been amazing. The descriptions of these remarkable works preludes the aims of this research as follows:

Al-Shatnawi [39] proposed a new method for hiding a secret message based on searching and finding identical bits between the secret messages and image pixel values. Jain et. al [40] proposed a method of using the edges of the images for hiding text messages in image Steganography. Bas et. al in [41] presented a methodology for designing and implementing highly undetectable stegosystems for real life digital media. This methodology allows payloads that are large in to remain undetected by steganalysis. In [42] researchers proposed an efficient steganalytic LSB matching based on image noise and experimental results show that even on low embedding ratios the detection accuracy can reach up to 79%. This also tells us that embedding payloads on image noise is not ideal and this is supported by Zhang et al in [43]. Hiding payloads on image noise is quite pejorative and may be subject to defeat with steganalysis. An Analysis of LSB and Discrete Cosine Transform (DCT) has been conducted by Walia and Jain in [44] and the analysis shows that DCT out performs LSB in terms of high ratios when evaluating using Peak Signal to Noise Ratio

(PSNR). This shall be ignored in this research since the approach is based on the spatial domain. Xu et. al [45] proposed an effective LSB based steganographic algorithm which is based on the classic K-means algorithm. A conducted test case shows that the algorithm can hide 60% of the size of the cover without any visual artefacts. Devi and Sharma [46] proposed an improved detection of LSB in gray scale and colour images. Their methods apply only for the LSB replacement method and not for LSB matching. This research focuses on the utilizing LSB insertion method based on matching RGB pixels as a better substitution for embedding, since this method is robust and also yields and retains high capacities for embedding. Taking these works into consideration, the aim is to design and implement steganographic algorithms that are efficient, robust and self-adaptive to multicore architectures.

2.7 Image Feature Extraction with Parallelism

Image feature extraction (IFE) is commonly used in many image processing applications. It is concerned with the extraction of features that are found on digital images which include colour, texture, shape and form. There are many applications where IFE is applied. Common applications include CBIR (Content Based Information Retrieval), Image Steganography applications and computer vision applications. All these applications rely on feature detection because the primary goal of IFE is met through feature detection. CBIR retrieves images by visible information such as colour, texture and shape for special types of images [8]. Human beings make use of feature detection on a daily basis to make sense of any object they come across. The amount of work to be done just for detecting the colour and the form in an image is huge and complex. A wide variety of image feature extraction algorithms exist since they are very handy. Having seen this, the focus of this research is narrowed entirely to optimizing the canny edge detector for Multicore Architectures.

2.8 Canny Edge Detector (CED)

A canny edge detector is an operator used commonly for image feature extraction and also adopted by many image processing algorithms. This operator involves the use of a multi-stage algorithm to detect a wide range of edges in images. Edge detection on its own is at the forefront of image processing and hence is crucial to have at an up-to scale level. Multicore Architectures are the next emerging technology to resolve performance issues on compute intensive problems.

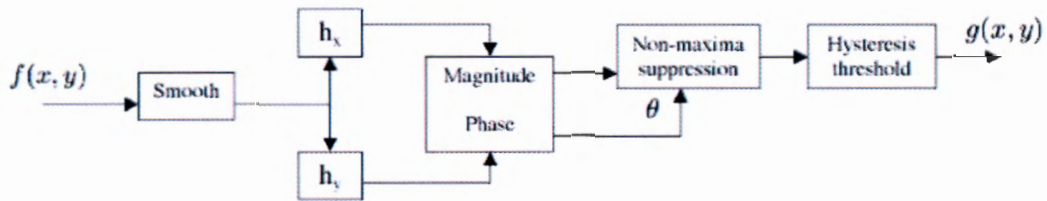


Figure 2.12: Canny Edge Detection Algorithm

Canny Edge Detector operator mainly aims at achieving:

- A. **Low error rate** - Reliable for accurate detection of only existent edges. For low error rates which yield good detection, canny edge detector uses Signal-To-Noise (SNR) ratio and its criterion for low error rates on detection is:

$$\text{SNR} = A \left| \int_{-T}^0 f(x) dx \right| / (\sigma \sqrt{\int_{-T}^T f^2(x) dx}) \quad (2.2)$$

B. **Good localization** - The distance between edge pixels detected and real edge pixels have to be minimized. The criterion for good localization is defined as:

$$L = 1 / \sqrt{\int_{-T}^T y^2 Pr(y) dy} \quad (2.3)$$

C. **Minimal response** - Restrict only one detector response per edge. In other words the detector should produce multiple maxima. According to Canny the minimal response criterion is defined by:

$$x_{max} f(x) = 2\pi \sqrt{\int_{-T}^T f'^2(x) dx} / \int_{-T}^T f'^2(x) dx \quad (2.4)$$

From the flow diagram seen in Figure 2.12 the CED algorithm can be represented as follows:

(a) **Filter out any noise**

Apply the Gaussian filter

(b) **Find the intensity gradient of the image**

Employ Sobel's Algorithm:

i) First apply a nominal pair of convolution masks (G_x , G_y)

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Compute the gradient strength and its direction

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$



(c) Now begin to apply Non-Maximum Suppression. This is to remove pixels that are not part of the edge

(d) Finally perform Hysteresis:

- i) If a pixel gradient is higher than the upper threshold, the pixel is accepted as an edge.
- ii) If a pixel gradient value is below the lower threshold, then it is rejected
- iii) If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the upper threshold.

2.8.1 Analytic review of CED algorithms

As illustrated by the authors in [8], IFE is a time consuming process and this is even worse when the images to be processed are in large quantities and of high quality. Consequently this problem is often encountered by image processing applications on the Internet because of the high frequency of image data on the Internet. Researchers in [9] have identified that the calculation of IFE algorithms constantly increases, and this contributes the most time consuming step in image steganography detection. Researchers in [8] discovered that most IFE methods do not pay much attention to performance and do not take note of the utilization of the highly developed microprocessor architectures. Almost all image processing applications are implemented serially, and this leads to poor results in terms of performance even on highly developed microprocessor architectures of the modern day computer systems. Researchers in [47] surveyed existing shape-based feature extraction. Yang and team in [47] recommended that efficient shape features must present essential properties such as identifiability, translation and noise resistance among others. They further outlined that a simple form of a shape descriptor is simply a set of numbers that describe a given shape feature, and in one of the requirements of a shape descriptor they state that the computation of distance between descriptors should be simple; otherwise execution time will cause overhead. Since the descriptor operates in serial and is not optimized for multicore architectures this may still be prevalent in applications making use of the descriptor. Hao and team in [48] successfully parallelized a Scale Invariant Feature Transform. In their work they state that in order to meet computation demands they optimized and parallelized SIFT to accelerate its performance on Multicore Architecture

systems. Furthermore they indicated that SIMD integrated with Multicore Architectures bring an extra 85% performance increase. Taking this into account, this research aims accurately to optimize and parallelize our algorithms so that they can meet computation demands and achieve best performance rates in terms of speedup. Zhang et. al [49] presented an improved parallel SIFT implementation which is able to process video images in real-time utilizing multicore processors; and the results showed great improvement in terms of speedup in comparison to GPU implementation. Cho and team in [50] spearheaded a study that construed the key factors used in the design and evaluation of image processing algorithms on massive parallel platforms. Clemons et. al [51] presented an embedded multicore design named EFFEX with novel functional units and memory architecture support capable of increasing performance on mobile vision applications while lowering power consumption rates.

2.9 Conclusion

In this chapter a strong theoretical perspective of Steganography and Image Feature Extraction algorithms was presented. This chapter also gave a strong background on Multicore Architectures and how they can be used as the new technology to intervene in performance problems associated with application of these algorithms. In brief, an analytical review with a systematic approach of the existing works proposed by other scholars was given. This has surely established a strong theoretical background on our research and this will enhance algorithm designs and implementation to be presented in the next chapter. In the next chapter a presentation of the implementation of the methodology is given.

Chapter 3

Model design, and algorithms implementation

3.1 Chapter Overview

In this chapter the design, approach and implementation of the model and algorithm are presented. A detailed explanation of each is given including the components, variables and configurations. In addition this chapter will present the research experimental setup which will demonstrate the implementation of the algorithm design and approach and how these work together to achieve the main goal.

3.2 Model Design and Approach

The preceding chapter presented the remarkable works which utilized multicore architectures as tools for solving compute intensive problems. The aim of this chapter is to provide a model with a novel design that relies on multicore architectures as the underlying technology to solve these problems.

The developed model takes a novel structural approach in which the design is unified with the problem. This model is not problem specific or domain specific, and hence can be applied to any real problem, which may belong to any particular domain, and this is for the sake of reusability and generality. The proposed model is composed of three layers as seen in Figure 3.1. These layers are the **Shell** (where real world problems are synthesized), the **Kernel** (which optimizes and restructures the problem in a parallel context suitable for the third layer) and lastly, the **Core** (where the problem is now synchronized and processed with multicore architectures).

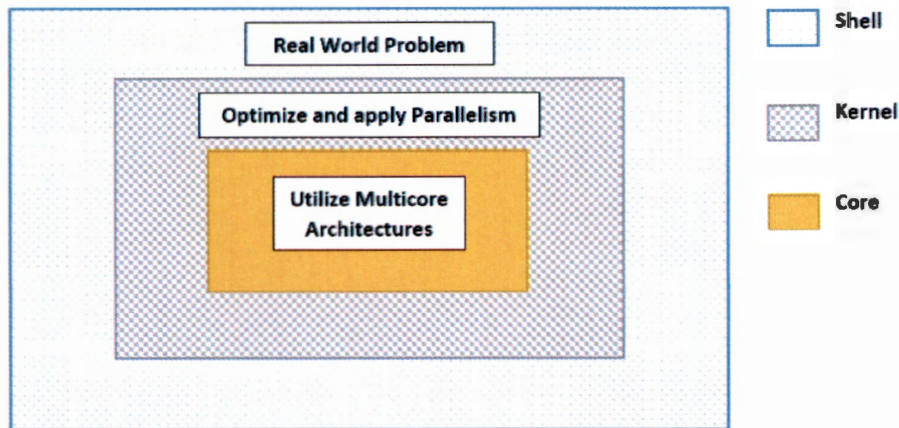


Figure 3.1: Node of the Proposed Model

A full description of the layers of the model is as follows:

3.2.1 The Shell

The shell mainly synthesizes any real world problem encountered and pragmatically processes it according to its specific domain. That is to say, the shell synthesizes the problem into an algorithm which relates with its domain. For example, if a particular problem falls within the web domain, or image processing domain then its algorithm will be processed in that particular context. Hence, the model applies related aspects of artificial intelligence in which the model as an agent learns which domain the problem falls into when synthesizing. The shell is mainly for defining a problem as an algorithm and identifying opportunities of parallelization for the kernel. When this layer has successfully executed, what is produced is a parallel algorithm that is now ready for the kernel.

3.2.2 The Kernel

The kernel plays a critical role in the model and hence is the most important layer. This layer optimizes the algorithm for the core layer which contains the underlying multicore architectures. This layer is aware of the underlying multicore architecture and hence optimizes according to it. For example, if the multicore architecture has 8 cores, the kernel will optimize the amount of work for each. The kernel consists of multiple steps in which each step specifically optimizes the algorithm and tunes it for the next phase.

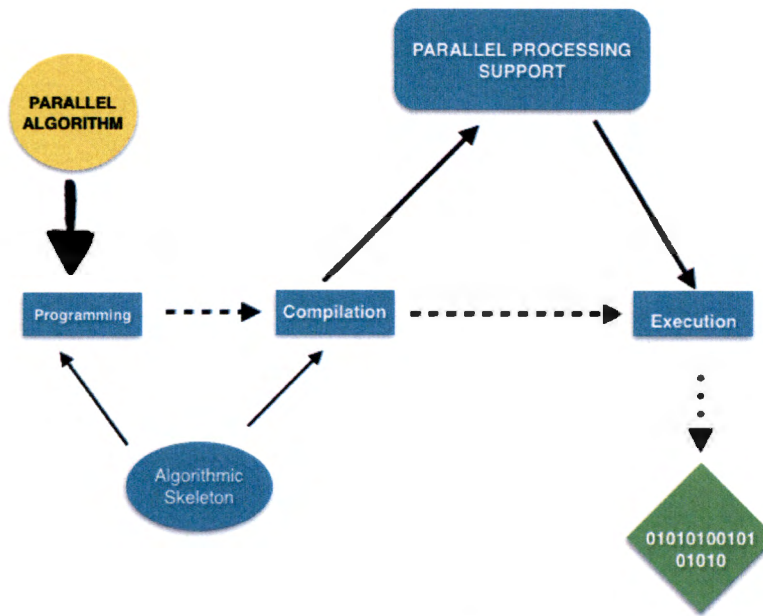


Figure 3.2: The Kernel Layer

There are three main phases of the kernel layer, namely programming, compilation, and execution, as can be seen in Figure 3.2. Each phase has interceptions which highly optimize the phases, and both the programming and compilation phases fully utilize algorithmic skeletons. Also, compilation yields parallel processing support, hence the binaries scheduled for processing are highly parallel and can take full advantage of the underlying multicore architecture.

3.2.3 The Core

The core mainly contains the multicore architecture of the system. However, this does not mean that the core is tied to any specific model. The core simply adapts to the multicore architecture within the system applying the model. If for example a multicore architecture is of two cores, as seen in Figure 3.3, then the kernel will optimize the problem for these and the cores will be mapped to the core of the model for execution. Hence, this model is not specific to any multicore architecture technology because new technologies evolve disruptively in the microprocessor industry. This model can still apply to any Multicore Architecture Technology.

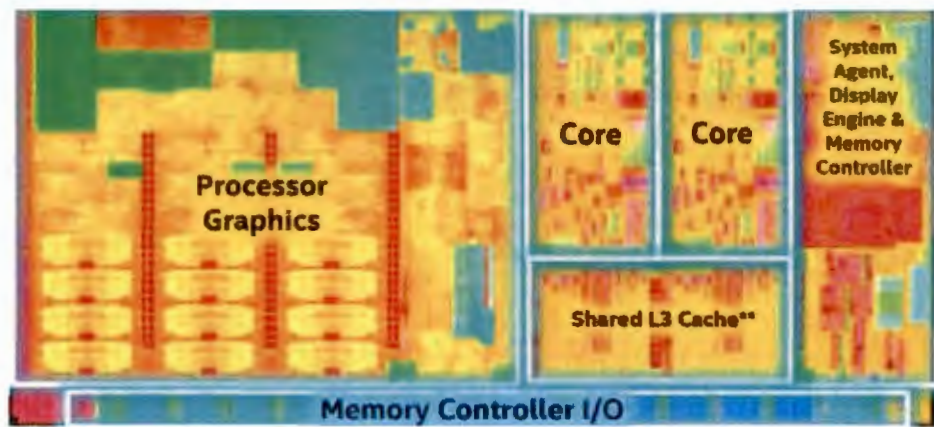


Figure 3.3: Dual Core Multicore Architecture

3.2.4 The Expanded Model

Since the developed model synthesizes real world problems, it has to be able to manage throughput because there can be many real problems. To cater for this need an expanded design of the model is proposed. This can allow these problems to be effectively handled in parallel. Because technology is always evolving depending on the consumer needs, the expanded design is tangent to the multicore architecture approach and allows structural management of computation.



Figure 3.4: Quad Node

Since there can be multiple real world problems occurring at the same time, an axiom which maintains the structural approach is proposed. The axiom affirms that for every four nodes of an entity in the model, there has to be a transformation. So for every four identical nodes of the same level of the model a batch shall be formed to produce a Quad node as seen in

Figure 3.4. Quad Nodes are more powerful than normal entities, hence it is recommended that they only should be used for high performance demand tasks and environments. Since performance demands get worse with time and at times there will be high demands, the axiom is applied to the Quad Node to form an Octal Node as seen in Figure 3.5.



Figure 3.5: Octal Node

The expanded design in summary simply follows this principle:

A. Synchronize real world problems such as:

- i) Streaming a video file
- ii) Computing Fibonacci sequence
- iii) Generating a hash table entry
- iv) Rendering a 3D image file

B. Apply the model. If there are more than three real world problems that are compute intensive:

- i) Form a Quad Node
- i) Form more Quad Nodes if needed but not more than four.

C. If there are more than three Quad Nodes required then,

- i) Form a Octal Node
- ii) Form more Octal Nodes if needed but not more than four.

D. End result

It is duly noted that any machine with two Octal Nodes will be sufficient for today's demands of computation and that they are very suitable for clusters in high performance computing (HPC) environments.

3.3 Model and Algorithm implementation

In this research a set of software tools for the experimental setup were used to implement the developed model and apply the algorithms. These are described briefly as follows:

3.3.1 Software Tools

Intel® Parallel Studio

Intel Parallel Studio provides a software development environment which enables developers to facilitate code development for parallel computing. Intel Parallel Studio provides a comprehensive environment which allows developers to design, optimize, analyse and inspect code for parallel programming. Since the multicore architecture vendor for the experimental setup is Intel®, Parallel Studio was chosen as the best software tool to provide full optimization of the developed parallel algorithms for multicore architectures.

The other reason Intel Parallel Studio was chosen is because it provides Intel® Cilk Plus tools. Cilk Plus provides multithreading and thus implements shared memory approach suitable for Multicore Architectures. Hence, Cilk Plus forms the basic foundation and key to the parallel implementations in this research. Cilk Plus guarantees composability and determinism by featuring a work stealing scheduler which assures that workers (cores) will not be idle during program execution. It allows developers to specify the order of parallelism which simply means that Cilk Plus gives researcher total full control of CPU parallelism at a high level.

Microsoft® Visual Studio

Microsoft Visual studio is an integrated development environment (IDE) which allows developers to fully develop applications that run on Microsoft Windows®, and it also allows developers to develop web services, websites and other software in different domains, such as image processing etc. In this research Microsoft Visual Studio was coupled with Intel Parallel Studio for a more advanced development environment that can allow full implementation of the parallel algorithms.

Open Source Computer Vision (OpenCV)

OpenCV is a comprehensive library available in different languages developed mainly for computer vision applications. Steganography and image feature extraction algorithms fall within the image processing domain, and hence OpenCV is the best library for the implementation. Hence, for the experimental setup, Visual Studio has been extended with OpenCV to cater for steganography and image feature extraction algorithms. The source code for implementing both algorithms is listed in **Appendix A** for Image Feature Extraction CED and **Appendix B** for Steganography.

3.3.2 Hardware setup

The hardware setup for the experiment consists of the following:

- A. CPU: Intel® Core™ i3-2130 CPU @3.4GHz with Hyper-Threading
- B. Topology: 2 physical cores and 4 logical cores
- C. Max Clock Speed: 3.40GHz
- D. Thermal Design Power: 65W

3.3.3 Experimental Setup and Application

Image feature extraction

For image feature extraction algorithm the CED algorithm mentioned in the previous chapter has been fully parallelized. Parallel patterns have been applied on the Gaussian filter and on Sobel's algorithm which finds the intensity gradient of the image. Abiding with Amdahl's law the hysteresis part of the CED algorithm has been noted to contribute

unparalleled work because of the serial elision it carries. CED algorithm was implemented on Microsoft Visual Studio, and the source code is listed in Appendix A. This algorithm was applied to different applications of different disciplines to see how it performs. The disciplines include Medical Research, Computer Vision, and Communication Research. For each these research areas, an application of CED algorithm has been carried out on both the optimized algorithm and the non-optimized algorithm, and the results were recorded for each.

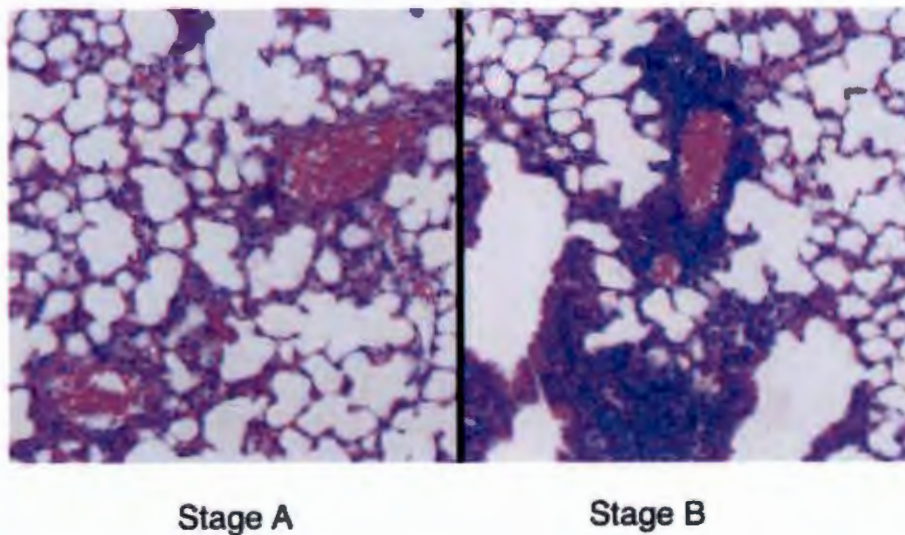


Figure 3.6: Medical Research Application

First an application of both optimized and non-optimized CED algorithm is shown in Figure 3.6. This application of the CED algorithm is useful for medical research applications and can be used for yielding different results depending on the applications viz. x-rays, Neuroimaging, Virus Mapping etc. After applying the CED algorithm using visual studio, the operator produces the feature extracted image seen in Figure 3.7.

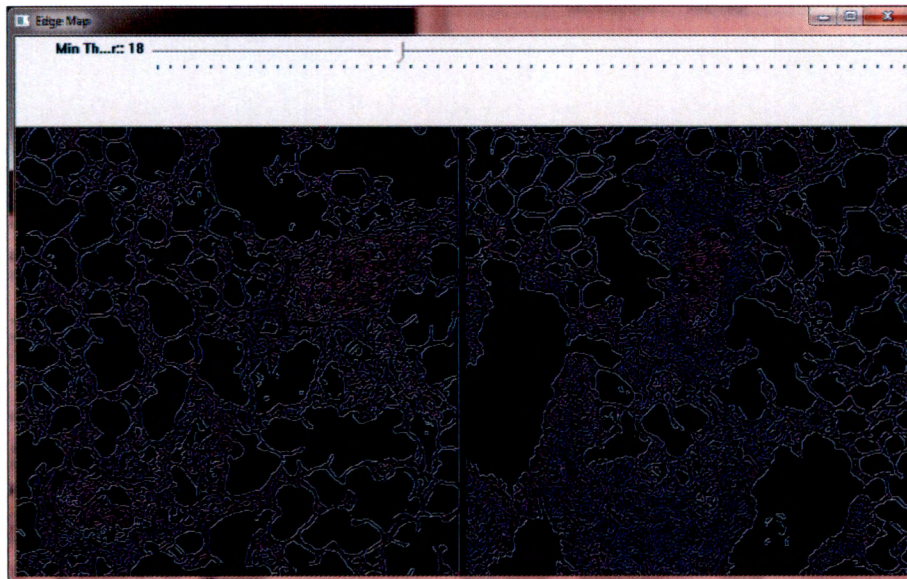


Figure 3.7: CED Applied on Medical Research Application

As can be seen in Figure 3.7, with the increase on the threshold the canny operator will filter pixel which are below average from the image just as stated in the hysteresis. As seen on Appendix A, line 22 and 23 the threshold is defined to be 100. The CED algorithm is highly suitable for medical research applications, and it can be instrumental in applications of virology which involve highly virulent viruses like the one shown on the micrograph in Figure 3.8.

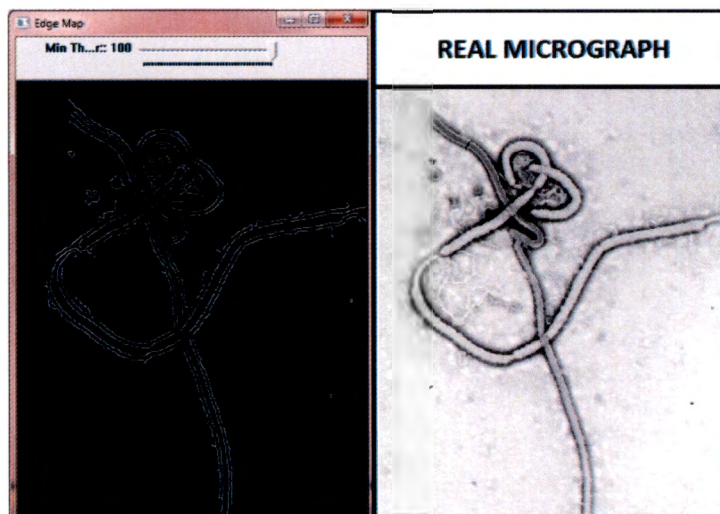


Figure 3.8: CED Applied on Other Medical Research Application

Applying CED algorithm on computer vision applications, the operator yields the output result seen in Figure 3.9. This would be very helpful in identifying different objects that the computer may encounter using computer vision.

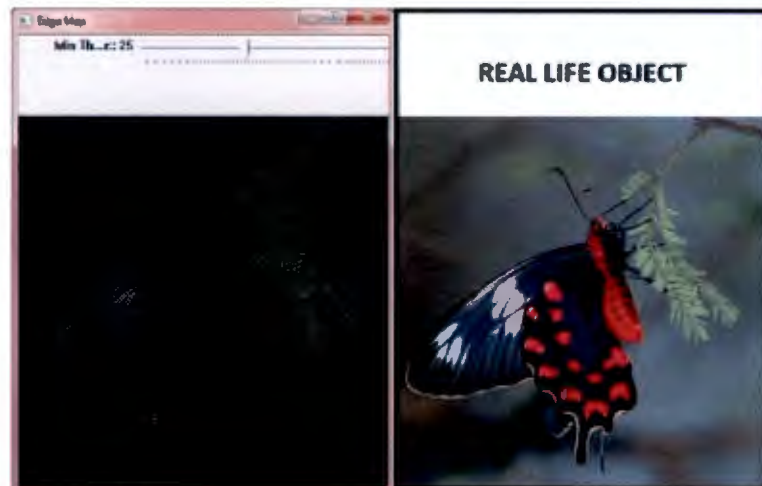


Figure 3.9: CED Applied on Computer Vision Research

For example, if the computer has a database containing characteristics, components and attributes of different objects, then it can scan and identify the object using the CED algorithm seen in figure 3.9 and figure 3.10 and from these figures it can be seen that the algorithm also performs well for computer vision applications.

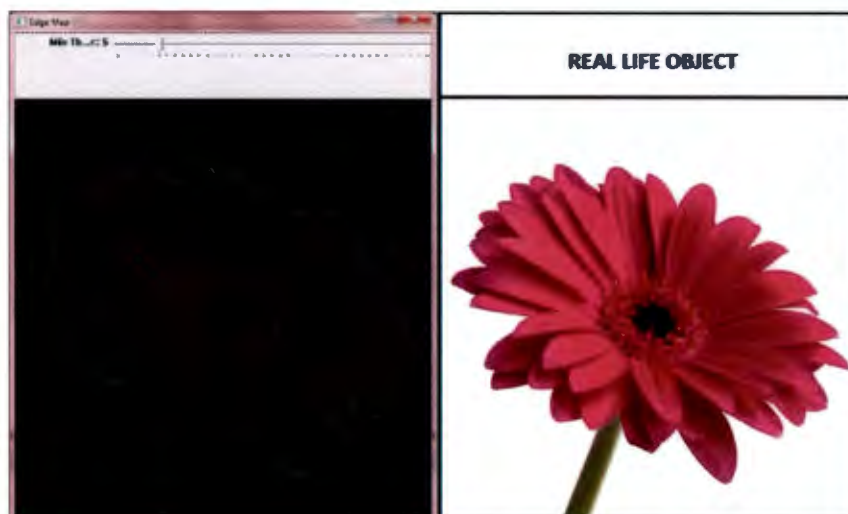


Figure 3.10: CED with Computer Vision

When applying the CED algorithm on Communication research application the operator yields the output result seen in Figure 3.11. The CED algorithm scans the signal extracts edges and it can come to the conclusion that the signal represents letter F in American Sign Language.



Figure 3.11: CED Applied on Communication Research

Steganography

For steganography the Least Significant Bit algorithm is applied on two images. Image A is used to hide Image B on its Least Significant Bits. The result is a Stegoimage (Image A embedded in Image B). The size of Image A has to be twice that of Image B. The LSB algorithm is applied on a GIS application of embedding a photographic satellite image on a topographic image as seen in Figure 3.12.

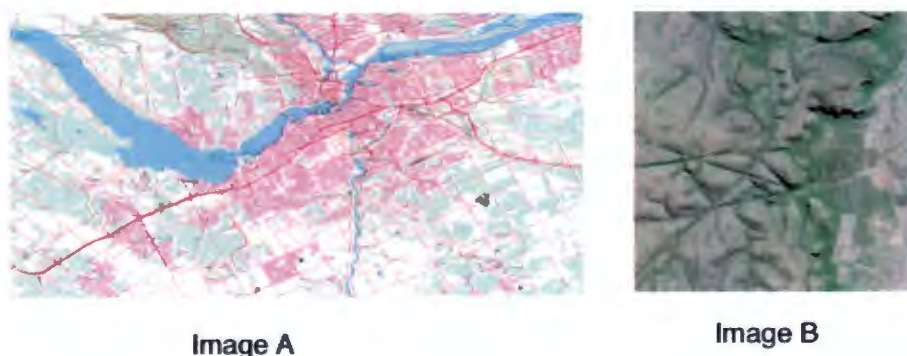


Figure 3.12: LSB Applied on GIS Research

An implementation of the developed LSB algorithm in Microsoft Visual Studio is applied to the images and the resultant output is displayed in Figure 3.13. Depending on the size and quality of the images the algorithm may take time processing the images. The source for this implementation is listed in Appendix B.

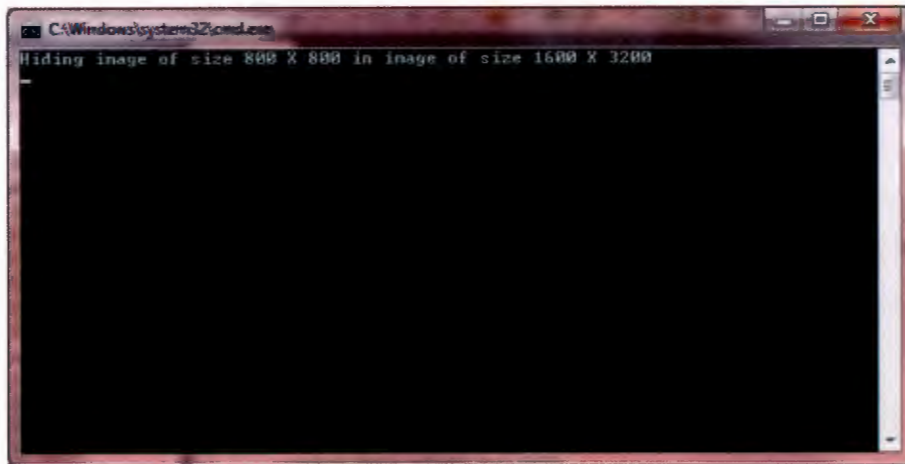


Figure 3.13: Screen Output for LSB Algorithm

For this implementation the screen output indicates that it is hiding an image of size 800x800 on an image of size 1600x3200. Then it outputs the Stegoimage which can be seen in Figure 3.14. It can be seen that the Stegoimage is identical to image A which is ideal for effective steganography.

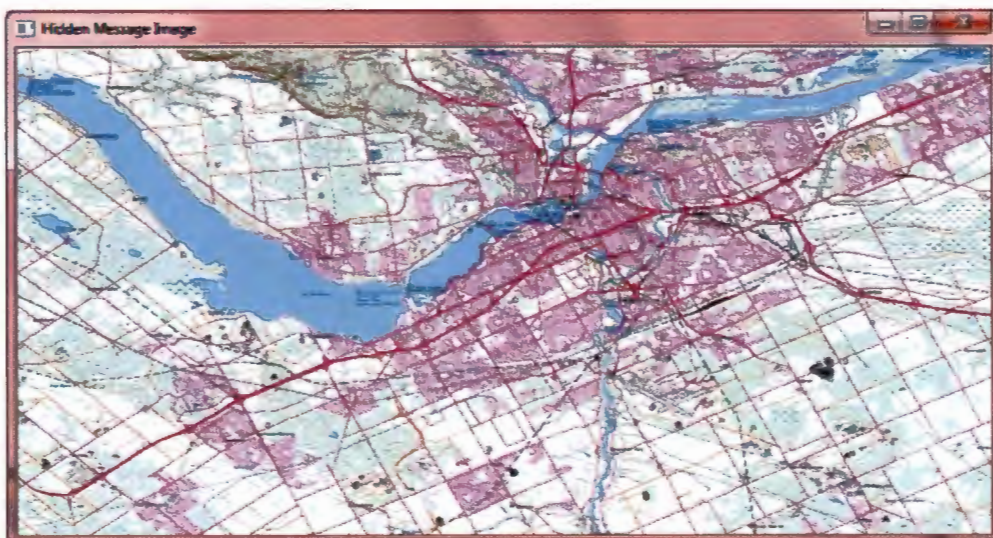


Figure 3.14: The Generated Stegoimage

3.4 Conclusion

This chapter presented the design and approach of the developed model and algorithms. The chapter demonstrated that the model is not limited and that the design can further be expanded to cater for high performance demanding environments. An axiom was also proposed which maintains the structural approach on the model. In addition the model was aligned with the proposed steganography and Image Feature Extraction algorithms by applying it to different research applications. For the Image Feature Extraction Algorithms the Canny Edge Detector was successfully setup and implemented using software tools from Intel®, Microsoft®, and the Open Source. CED was applied to Communication research, Medical Research, and Computer Vision Research and it was shown that it yielded good results. Lastly an implementation of the Least Significant Bit algorithm was presented and it was able to successfully hide an image within an image using the parallel approach provided by the developed model. In the next chapter a discussion of the obtained results from the experiments conducted shall be given.

Chapter 4

Model results and discussion

4.1 Chapter Overview

This chapter shall present the results obtained from the experiments conducted in the previous chapter. This presentation is also coupled with a detailed discussion of the results obtained from both algorithms. Finally this chapter will provide a critical analysis on the performance of the developed algorithms.

4.2 Image Feature Extraction Results

The following results were obtained from experiments conducted on the Canny Edge Detector.

4.2.1 CPU Sampling and Total Usage



Suboptimal Implementation

The sampling method used collects profiling data for every 10 million processor cycles and this is very useful for detecting performance issues. For the suboptimal or non-optimal implementation of CED algorithm the profiler collected 8,992 samples. A graphical representation of CPU usage over wall clock time in seconds is shown in Figure 4.1. From this graph a low CPU usage can be seen which is not ideal for application performance and user experience. The low CPU usage is in response to the total sample count.

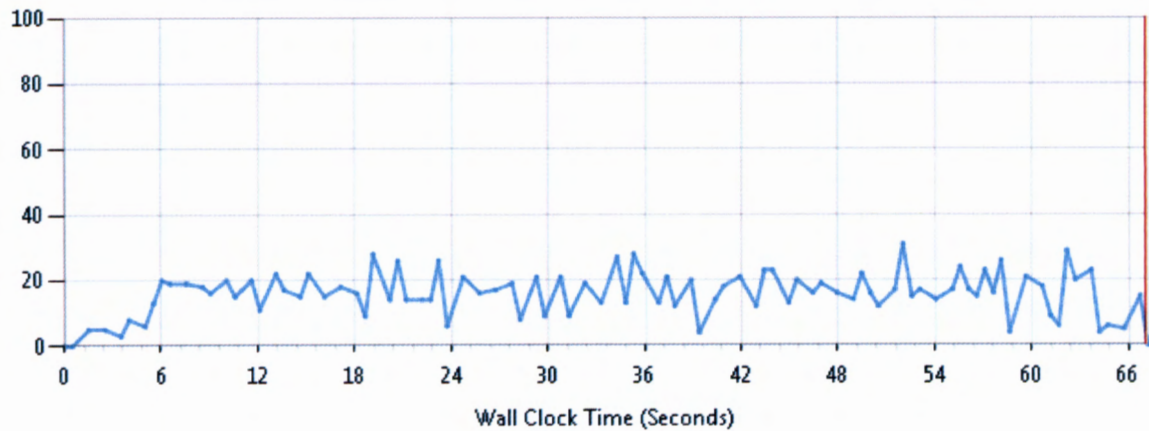


Figure 4.1: Suboptimal CPU Usage over Wall Clock Time

Optimal Implementation

For the optimal implementation of the CED algorithm the profiler collected 34,884 samples. In Figure 4.2 the graphical representation of CPU usage over wall clock time in seconds can be seen for the fully optimized CED algorithm. This graph shows better CPU usage which is ideal for application performance and user experience. The efficient CPU usage is also in response to the total sample count.

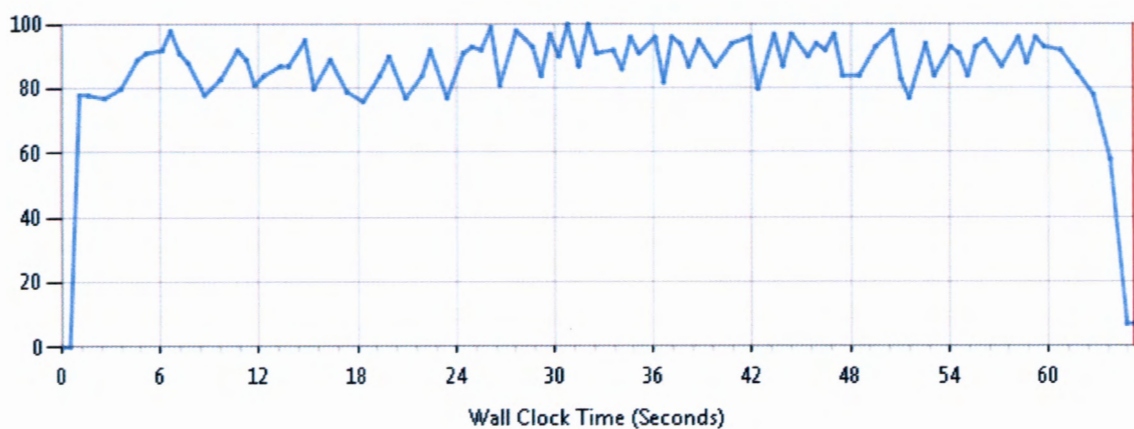


Figure 4.2: Optimal CPU Usage over Wall Clock Time

From the observed results it can be seen that the optimized CED algorithm outperforms the suboptimal implementation of the CED algorithm. The above results provide only information for total CPU Usage in percentage. To be precise the total usage per core must be observed.

4.2.2 Total CPU Usage per Core

Suboptimal Implementation

To be precise, if the CED algorithm is stable and durable in terms of performance, total usage per core must be observed. In Figure 4.3 the total CPU usage per core for the suboptimal implementation can be seen precisely. From the figure it can be seen that the utilization is uneven, meaning that some cores may be idle while others are working. This is not ideal for overall application performance on a Multicore Architecture system.

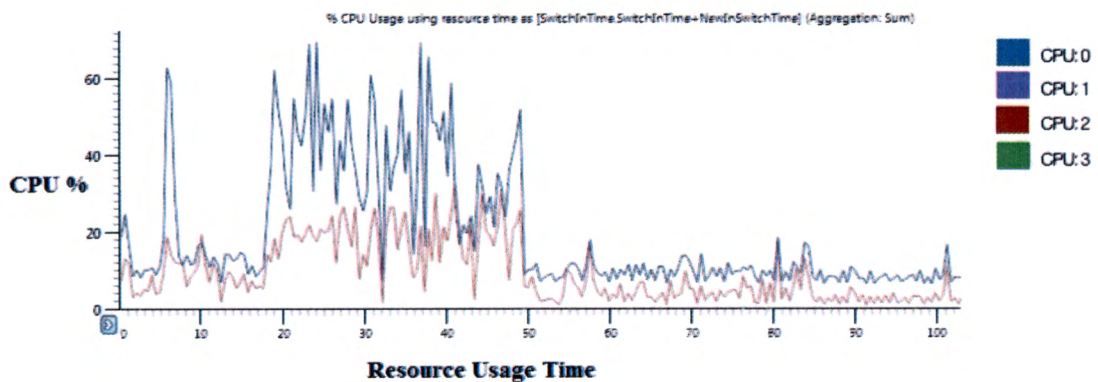


Figure 4.3: Suboptimal CPU Usage Per Core

Optimal Implementation

The total usage per core for the optimal implementation of the CED algorithm is shown in Figure 4.4 and it can be seen that there is an ideal usage per core. It is observed that utilization per core is even and well distributed among the cores. This is mainly because of the work stealing scheduler provided by Cilk Plus. This even distribution is ideal for Multicore Architecture based systems and will enhance overall application performance and user experience.

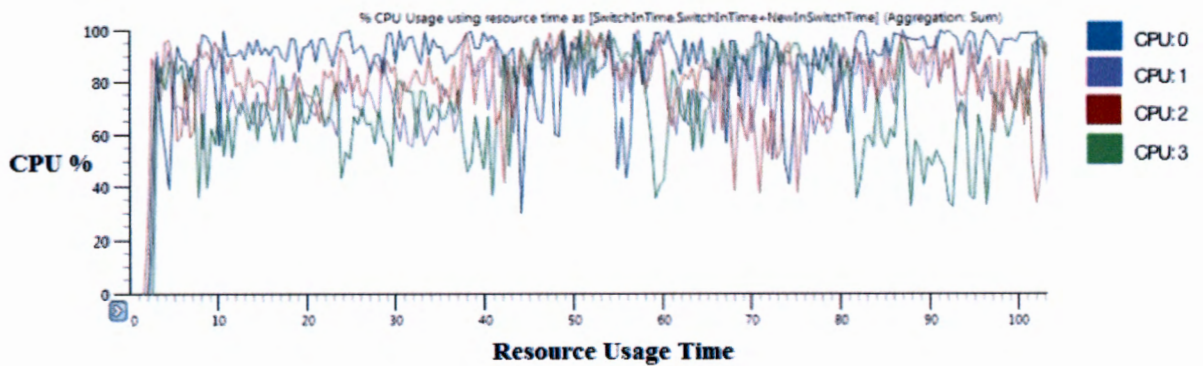


Figure 4.4: Optimal CPU Usage per Core

4.2.3 Context Switch per Core

Suboptimal Implementation

Context switches are essential because they can affect the performance of Multicore Architecture systems. Context switches occur when the operating system kernel switches the CPU between threads. Hence, in a Multicore Architecture system if the kernel only switches one core out of four between threads it can increase competition for CPU and raise tension which can degrade overall performance of the system. In Figure 4.5 it can be seen that the kernel switches only two cores out of four, and hence this explains why there are high context switch rates which may create tension and degrade overall system performance.

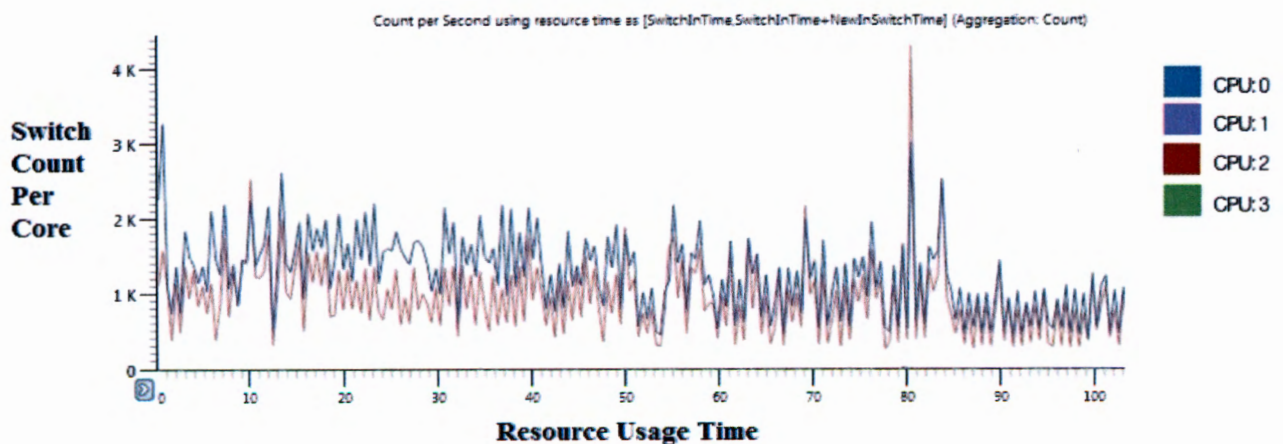


Figure 4.5: Suboptimal Context Switch per Core

Optimal Implementation

Context switches count for the optimal implementation of the algorithm as can be seen in Figure 4.6. It is evident from the figure that the kernel switches the CPU at a full swing of 4 out of 4. In addition to this, it is evident that the switch rate is relatively low, and hence it does not compromise the performance of the Multicore Architecture system. However, there are occasional high peaks displayed on the graph in Figure 4.6. These occasional high peaks could be signalling an emergence of a bug or overhead. In this case an analysis is essential to rectify such occasional peaks. Other than that from this graph an ideal Context Switch for a Multicore Architecture system can be seen.

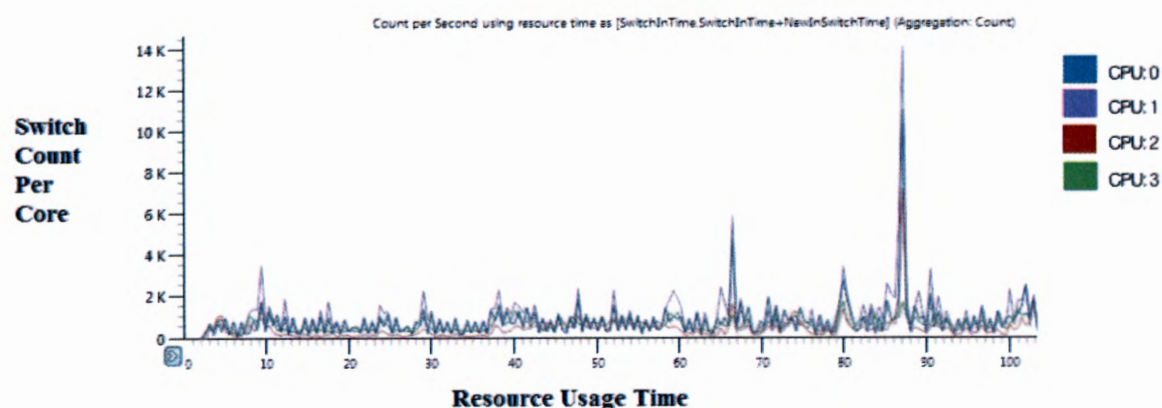


Figure 4.6: Optimal Context Switch per Core

4.3 Steganography Results

From the experiments conducted on the Least Significant Bits Algorithm the following has been measure and observed:

4.3.1 CPU Sampling and Total Usage

Suboptimal Implementation

The same CPU sampling method which collects profiling data for every 10,000,000 processor cycles has been used. This will help to evaluate the algorithm of performance issues. The total number of samples collected by the profiler for the suboptimal implementation of the LBS algorithm is 433 samples. Figure 4.7 shows the graphical representation of the CPU usage over wall clock time in seconds. This graph shows poor CPU usage over wall clock time.

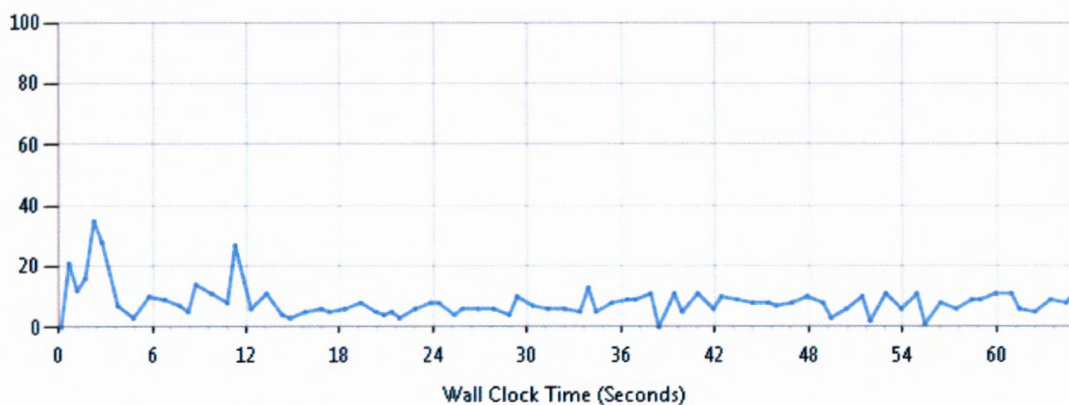


Figure 4.7: Suboptimal CPU Usage over Wall Clock Time

Optimal Implementation

For the optimal implementation of the LSB algorithm the profiler collected 32,642 samples. In Figure 4.8 a graphical representation of the CPU usage over wall clock time in seconds is shown. This graph shows better CPU usage over wall clock time which indicates efficiency and good overall performance, which is ideal for user experience.

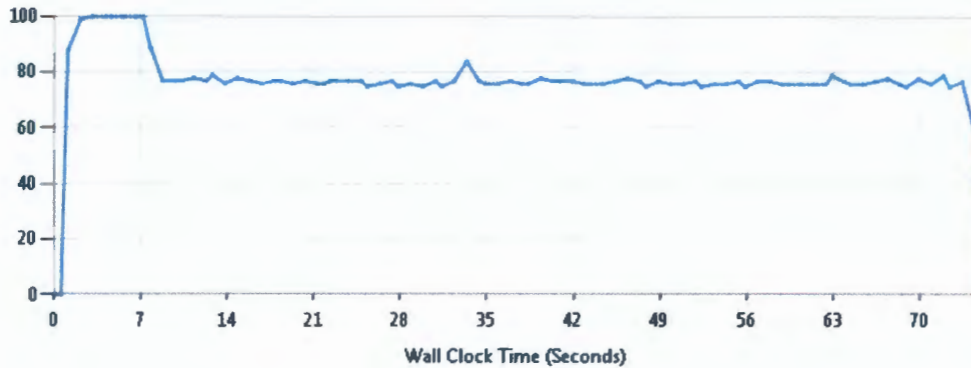


Figure 4.8: Optimal CPU Usage over Wall Clock Time

4.3.2 Total CPU Usage per Core

Suboptimal Implementation

To confirm the efficiency of the LSB algorithm an observation of the total usage per core by the steganography algorithm is conducted. Figure 4.9 shows us precisely the total CPU usage per core for the suboptimal implementation. This graph indicates that the algorithm does not efficiently utilize the CPU cores, which is not ideal for performance. This also tells us that this algorithm is not suitable for multicore architectures.

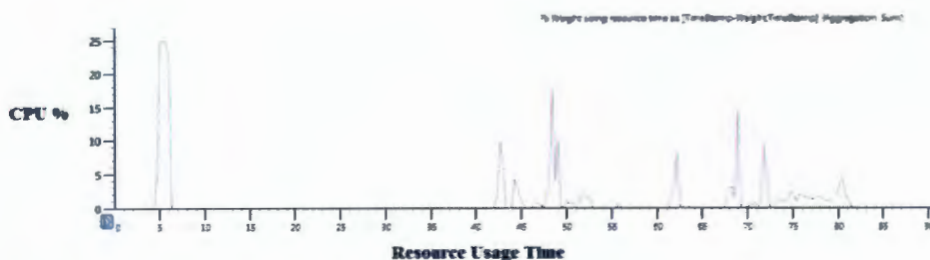


Figure 4.9: Suboptimal CPU Usage per Core

Optimal Implementation

The total usage per core for the optimal implementation of the steganography algorithm is shown Figure 4.10 and the graph shows an ideal usage per core. Utilization per core is precisely even and well distributed among the cores. This confirms the efficiency of the algorithm and shows that the optimized LSB algorithm is suitable and performs well on multicore architecture systems.

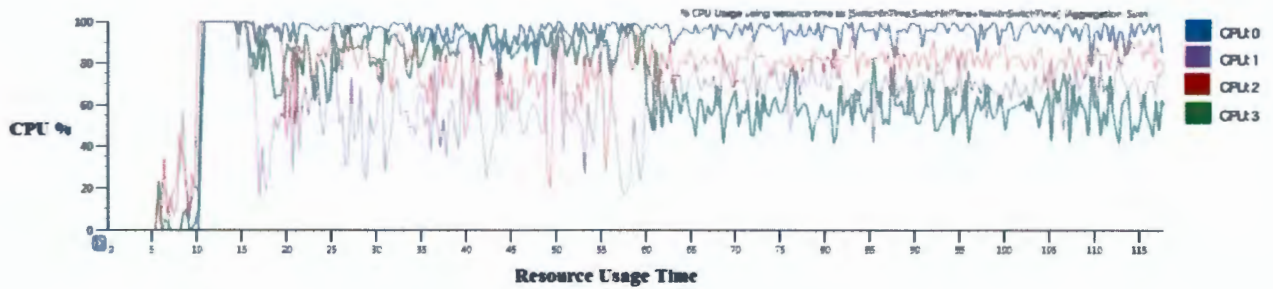


Figure 4.10: Optimal CPU Usage per Core

4.3.3 CPU Utilization by Process

Suboptimal Implementation

Monitoring CPU usage by the created process when the algorithm is in execution is important for reporting performance. In Figure 4.11 it can be seen that the overall CPU usage for the suboptimal implementation is poor and not consistent.

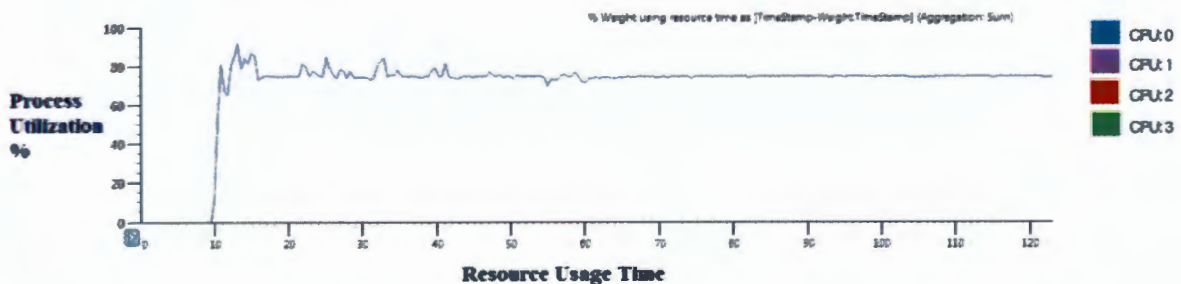


Figure 4.11: Suboptimal CPU Usage by Process

Optimal Implementation

For the optimal implementation Figure 4.12 shows a better CPU usage that is almost three times better than the suboptimal implementation. The graph in Figure 4.11 also shows a fairly consistent usage which is ideal for application performance.

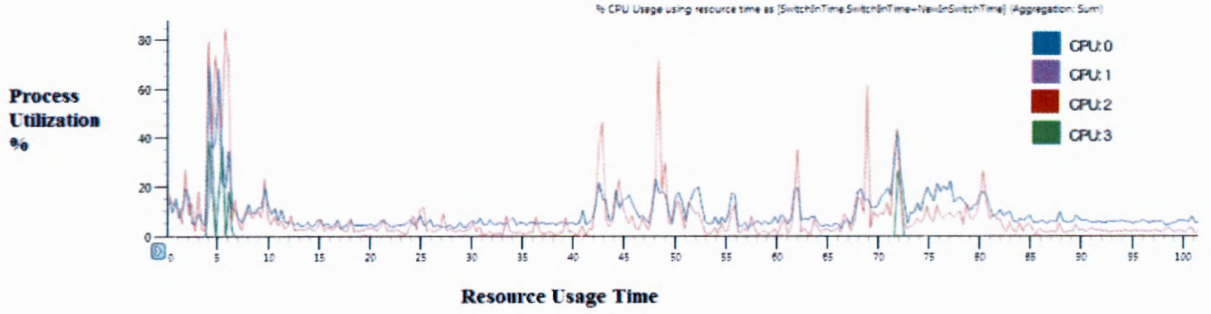


Figure 4.12: Optimal CPU Usage by Process

The above figures successfully reported the results obtained from the experiments. These results show great improvements in performance and stability and serve as proof of functionality of the developed model and algorithms. In the next section an analysis of the developed algorithm is presented.

4.5 Critical Analysis

In this section an analysis of the developed parallel algorithms for CED and LSB is provided. Since the aim is to provide parallelism for multicore architecture systems, the research focus will be to pay close attention to Amdahl's law also known as the fundamental principle of parallelism because it does hold for multicore architectures. Because of Gustafson's law, it is recommended that for improved performance on a large scale system with more cores, the rate of work for the algorithms be increased so that the performance of the algorithms can be able to scale with core count:

$$\text{Speedup} = \frac{1}{(1-f) + (\frac{f}{n})} \quad (4.1)$$

A closer look into Gustafson's law is also taken into consideration. This law states that computation's serial fraction does not necessarily scale with large data sets and hence as the number of processors increases, the problem size must also increase for scaled speedup.

$$\text{Speedup} = P - a * (P - 1) \quad (4.2)$$

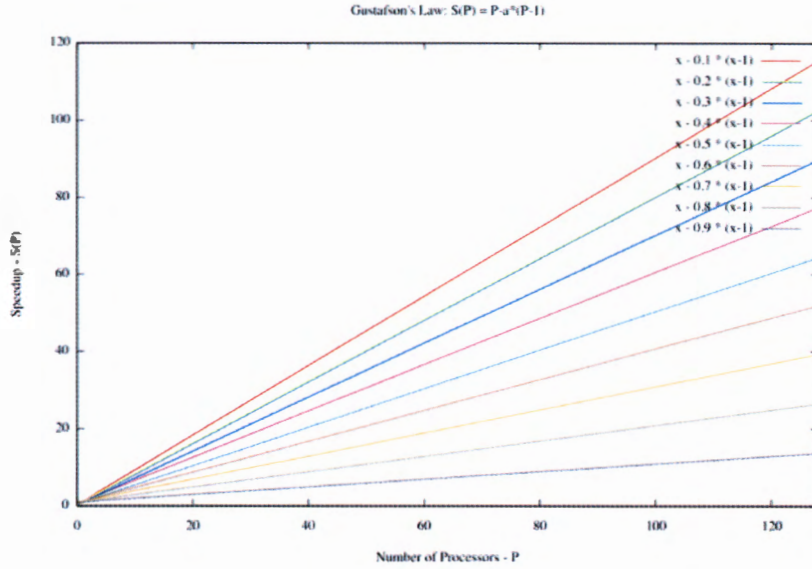


Figure 4.13: Gustafson's Law [1]

Taking these laws into account, the analysis of the developed algorithms is provided. For the analysis, image quality metrics together with other parallel computing parameters such as execution rate, synchronization, pre-emption and idle time or sleep rate were used. The parallel computing parameters are shown in Figure 4.14 and Figure 4.15. The Mean Squared Error (MSE) metric is used to evaluate the IFE algorithm. MSE gives cumulative squared error between the original image and the reproduced image. The aim is to achieve efficient feature extraction with reduced MSE rates.

Mean-Squared Error (MSE):

$$MSE = \frac{\sum_{j=1}^N (\sum_{i=0}^M (X_{i,j} - Y_{i,j})^2)}{MN} \quad (4.3)$$

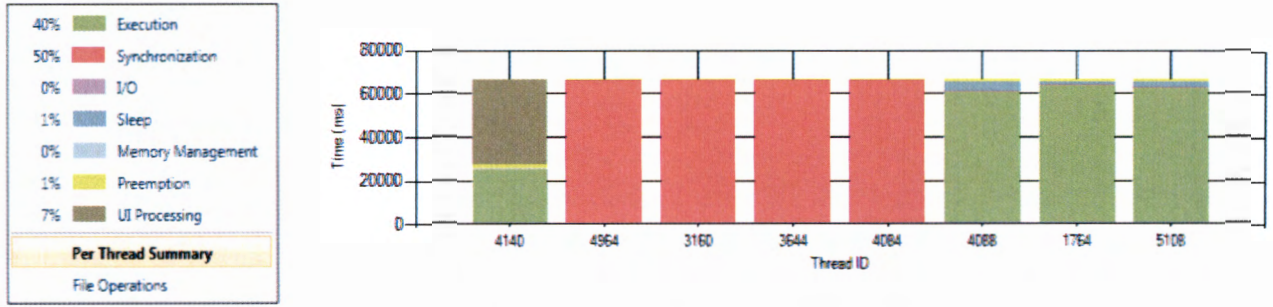


Figure 4.14: IFE Profiling per Thread

Furthermore, Peak Signal Noise to Ratio (PSNR) metric is used to evaluate the image quality of the reproduced Stegoimage file by the steganography algorithm. PSNR requires that first, the value of MSE be calculated. Comparing the thread summary reports in Figure 4.14 and Figure 4.15, it can be seen that the rates of the algorithms vary and this is mainly because each algorithm applies certain forms of parallelism at different times in a different manner (approach). However, the basic principle remains the same behind these two algorithms. Therefore for IFE, it can be seen that it has less execution time with high synchronization rates in contrast to the Steganography algorithm. This is because Parallel execution is more complex on the IFE algorithm than in Steganography algorithm and this results in high synchronization rates. The rest of these values are summarized in Table 4.1.

Peak Signal-to-Noise Ratio (PSNR):

$$PSNR = 10 \log_{10} \frac{R^2}{MSE} \quad (4.4)$$

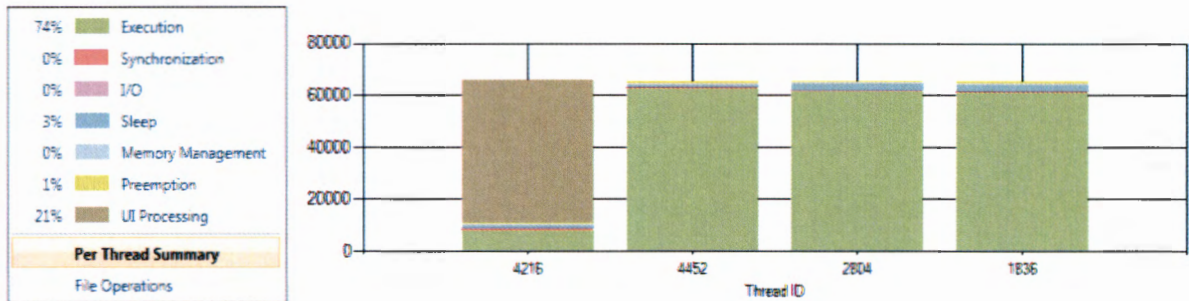


Figure 4.15: Steganography Profiling per Thread

Table 4.1: Metric Summary Comparison for IFE and Steganography

Metric	IFE	Steganography
Execution	40%	70%
Synchronization	50%	0%
IO	0%	0%
Sleep	1%	3%
Memory Management	0%	0%
Preemption	1%	1%
UI Processing	7%	21%

4.6 Conclusion

This chapter provided a comprehensive discussion of the results obtained from experiments conducted. The results show that the developed parallel algorithms are more effective and efficient than the normal algorithm. This confirms that the algorithms are suitable and perform well on Multicore Architectures. Lastly, this chapter provided a critical analysis on the developed parallel algorithm in terms of the MSE, PSNR and parallel computing parameters.

Chapter 5

Summary, conclusions and future work

5.1 Chapter Overview

This chapter provides a summary of this research work. In brief, this chapter will give a summary of the key points of the research. This is followed by the conclusion of the research work, and lastly this chapter discusses relevant future work.

5.2 Summary

Multicore Architectures bring enhanced empowering technology for modern computer systems. These systems at large remain to be the core key drivers of technology, as Moore's law continues to dictate technological realms. In the 21st century personal computers are the basic technology empowering people. Multicore Architectures now dominate personal computers, and this means that almost all personal computers today are parallel.

The parallel feature created by the presence of Multicore Architectures on personal computers presents a need for parallel programming to enhance the development of software in the future. In particular, compute intensive algorithms such as the one for steganography and Image Feature Extraction need to be highly optimized to fully take advantage of the prevalent technology of Multicore Architectures, since their usage is in high demand due to the apparent status of development on the Internet and the social media realm.

Addressed in this research work, is how these high demands can be met on the ubiquitous and heterogeneous platforms of Multicore Architectures. This research began by stating the critical problem statement identified by scholars and researchers. The majority of the studies cited indicated that most algorithms developed lacked parallelism which is a necessity for effective execution on Multicore Architecture systems.

The goal of this research was to successfully provide Novel and Smart parallel algorithms in Steganography and Image Feature Extraction that are optimized to obtain enhanced performance in Multicore Architectures. Through the proposed model we were able to impart some intelligence to our algorithm by involving an axiom. The model employed an axiom for maintaining structured parallelism. The developed algorithms gained novelty because, unlike traditional algorithms which fail to scale well on multicore architectures, the results in this research showed that they are highly suitable for Multicore Architectures.

The summary by chapters is as follows:

- i) **Chapter One** - Introduced the study of interest in this research work followed by a clear research plan on how the goals of this research would be met.
- ii) **Chapter Two** - Provided a full literature review on the area of interest which was beneficial for the approach and execution of the research work.
- iii) **Chapter Three** - Provided designs and implementation of the model and both the algorithms needed to achieve the main goal of the research.
- iv) **Chapter Four** - Discussed the results obtained from the experiments conducted and concluded with critical analysis of the results.
- v) **Chapter Five** – Provided a summary, conclusion and ideas for future work.
- vi) **Appendix A** – Provides source code for Image Feature Extraction application
- vii) **Appendix B** – Provides source code for Steganography application

5.3 Conclusion

In conclusion, this research reports the development and establishment of Steganography and Image Feature Extraction algorithms that scale well in Multicore Architecture platforms. Image processing algorithms are highly mathematical by nature, hence they demand high computation. The proposed model can be used to optimize any algorithm that needs to scale well in Multicore Architectures. This research work showed that the model successfully optimized the algorithms to be effective in Multicore Architectures.

The results obtained during the experiments as an implementation of the model and algorithms developed indicated good speed up rates and compliance with Amdahl and Gustafson's law which are part of the design principles of Multicore Architectures. The results showed high outputs as compared to ordinary algorithms, which could only span about 433 samples, while the developed algorithm using the proposed model approach obtained 32,642 samples. From the observations documented in this research work we can conclude that Multicore Architecture Optimization is a necessity for the scaled performance of Steganography and Image Feature Extraction algorithms.

5.4 Future Work

In future, the model developed in this research could be applied to the different problem domains of different platforms. The model must be evaluated and validated so that it can be a solution to the prevalent failures of software when scaling on Multicore Architecture systems. The main aim of the future work is to establish the model as the standard design principle for the development of algorithms and applications for Multicore Architecture Computer Systems.

Appendix A

// CED.cpp : Defines the entry point for the console application.

// Source code for the CED program

//#include "stdafx.h"

//#include "opencv2/imgproc/imgproc.hpp"

//#include "opencv2/highgui/highgui.hpp"

#include<opencv\highgui.h>

#include<opencv2\imgproc\imgproc.hpp>

#include <stdlib.h>

#include <stdio.h>

//#include <cilk/cilk.h>

using namespace cv;

/// Global variables

Mat src, src2, src_gray, src_gray2;

Mat dst, detected_edges;

Mat dst2, detected_edges2;

int edgeThresh = 1;

int lowThreshold;

int lowThreshold2;

int const max_lowThreshold = 100;

int const max_lowThreshold2 = 100;

int ratio = 3;

```

int ratio2 = 3;

int kernel_size = 3;

int kernel_size2 = 3;

char* window_name = "Edge Mappings";

char* window_name2 = "Edge Map";


/**
 * @function CannyThreshold
 * @brief Tracker callback - Canny thresholds input with a ratio 1:3
 */
void CannyThreshold(int, void*)
{
    /// Reduce noise with a kernel 3x3
    blur( src_gray, detected_edges, Size(3,3) );

    /// Canny detector
    Canny( detected_edges, detected_edges, lowThreshold, lowThreshold*ratio, kernel_size );

    /// Using Canny's output as a mask, we display our result
    dst = Scalar::all(0);

    src.copyTo( dst, detected_edges);

    imshow( window_name, dst );

}

void CannyThresholder(int, void*)
{
    /// Reduce noise with a kernel 3x3
    blur( src_gray2, detected_edges2, Size(3,3) );

```

```

/// Canny detector

Canny( detected_edges2, detected_edges2, lowThreshold2, lowThreshold2*ratio2, kernel_size2 );


/// Using Canny's output as a mask, we display our result

dst2 = Scalar::all(0);


src2.copyTo( dst2, detected_edges2);

imshow( window_name2, dst2 );

}


/** @function main */

int main( int argc, char** argv )
{

/// Load an image

//src = imread( argv[1] );

//src = imread("C:/F2.jpg", CV_LOAD_IMAGE_COLOR);

src2 = imread("C:/smiley.jpg", CV_LOAD_IMAGE_COLOR);


//Mat src1;

//src1 = imread("lena.jpg", CV_LOAD_IMAGE_COLOR);

//namedWindow( "Original image", CV_WINDOW_AUTOSIZE );

//imshow( "Original image", src1 );


//if( !src.data )

//{ return -1; }


if( !src2.data )

{ return -1; }

```

```

/// Create a matrix of the same type and size as src (for dst)

//dst.create( src.size(), src.type() );

dst.create( src2.size(), src2.type() );


/// Convert the image to grayscale

//cvtColor( src, src_gray, CV_BGR2GRAY );

cvtColor( src2, src_gray2, CV_BGR2GRAY );


/// Create a window

//namedWindow( window_name, CV_WINDOW_AUTOSIZE );

namedWindow( window_name2, CV_WINDOW_AUTOSIZE );


/// Create a Trackbar for user to enter threshold

//createTrackbar( "Min Threshold:", window_name, &lowThreshold, max_lowThreshold,
CannyThreshold );

createTrackbar( "Min Thresholder:", window_name2, &lowThreshold2, max_lowThreshold2,
CannyThresholder);


/// Show the image

//CannyThreshold(0, 0);

CannyThresholder(0, 0);


/// Show the image

//CannyThreshold(0, 0);


//cilk_sync;


/// Wait until user exit program by pressing a key

waitKey(0);

return 0;

```

Appendix B

// Steganography.cpp : Defines the entry point for the console application.

// Source code for the Steganography program

#include <stdio.h>

#include "opencv2/imgproc/imgproc.hpp"

#include <opencv/highgui.h>

#include <opencv/cv.h>

#include <stdlib.h>

#include <cilk/cilk.h>

#define EXTRACT_BIT(n,i) ((n&(1<<i))>0)

using namespace std;

using namespace cv;

int main(int argc, char **argv)

{

IplImage *input; // input image

IplImage *hide; // image to hide

IplImage *output; // output image

uchar *data;

uchar *inputData;

uchar *hideData;

int messageChar;

int i = 0, j = 0, dataPos = 48;

int height, width, nchannels;

int hideHeight, hideWidth, hideNchannels;

/*if (argc < 3)

```

{

printf("Usage : %s <C:/AlphaStegos.jpg> <C:/BetaStegos.jpg>\n", argv[0]);

exit(0);

}*/

//src = imread("C:/AlphaStegos", CV_LOAD_IMAGE_COLOR);

//input = cvLoadImage(argv[1], CV_LOAD_IMAGE_COLOR);

//hide = cvLoadImage(argv[2], CV_LOAD_IMAGE_COLOR);


input = cvLoadImage("BetaStegos.jpg", CV_LOAD_IMAGE_COLOR);

hide = cvLoadImage("AlphaStegos.jpg", CV_LOAD_IMAGE_COLOR);


//input image height and width

height = input->height;

width = input->width;

nchannels = input->nChannels;

/*

*Use 24 bits (1 bit per pixel) to save hide image height and width in output image

*/

hideHeight = hide->height;

hideWidth = hide->width;

hideNchannels = hide->nChannels;

/*

if (height*width < (hideHeight*hideWidth * 8 + 48))

{

printf("Please input source image of larger size\n");

exit(0);

}*/

if (nchannels != 3 || hideNchannels != 3)

{

printf("Sorry currently only support for RGB images\n");

```

```

        exit(0);
    }

    printf("Hiding image of size %d X %d in image of size %d X %d\n", hideHeight, hideWidth,
    height, width);

    //make an empty image with the same width and height as that of input image
    output = cilk_spawn cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 3);

    data = (uchar *)output->imageData;

    inputData = (uchar *)input->imageData;

    hideData = (uchar *)hide->imageData;

    //save the height and width in output image 24 pixels for height and 24 pixels for width

    //save height
    cilk_for(i = 0; i<24; i++)
    {
        data[i] = inputData[i] & 254;

        data[i] = data[i] + EXTRACT_BIT(hideHeight, i);
    }

    //save width
    for (i = 0; i<24; i++)
    {
        data[i + 24] = inputData[i + 24] & 254;

        data[i + 24] = data[i + 24] + EXTRACT_BIT(hideWidth, i);
    }

    //start saving the image from the 48th pixel
    for (i = 0; i<hideHeight*hideWidth * 3; i++)
    {
        messageChar = hideData[i];

        //printf("%d\n", messageChar);

        cilk_for(j = 0; j<8; j++)
        {
            data[dataPos] = inputData[dataPos] & 254;

```

```

        data[dataPos] = data[dataPos] + EXTRACT_BIT(messageChar, j);

        //printf("%d\t",data[dataPos]);

        dataPos++;

    }

    //printf("\n");

}

//hiding image completes

//copy the rest of the image to the output image

cilk_for(j = dataPos; j<height*width * 3; j++)

{

    data[j] = inputData[j];

}

//save the image in bmp or png format because they are lossless compression

cvSaveImage("stegHideImage.png", output, 0);

cvNamedWindow("Original Image", CV_WINDOW_AUTOSIZE);

cvNamedWindow("Hidden Message Image", CV_WINDOW_AUTOSIZE);

cvShowImage("Original Image", input);

cvShowImage("Hidden Message Image", output);

cilk_sync;

cvWaitKey(0);

cvReleaseImage(&input);

cvReleaseImage(&output);

cvReleaseImage(&hide);

cvDestroyWindow("Original Image");

cvDestroyWindow("Hidden Message Image");

return 0;

}

```

References

- [1] Michael McCool, James Reinders, and Arch Robison. *Structured parallel programming: patterns for efficient computation*. Elsevier, 2012.
- [2] Dinesh Agarwal et al. “Acceleration of bilateral filtering algorithm for manycore and multicore architectures”. In: *Parallel Processing (ICPP), 2012 41st International Conference on*. IEEE. 2012, pp. 78–87.
- [3] Peter Pacheco. *An introduction to parallel programming*. Elsevier, 2011.
- [4] Maninder Singh Rana, Bhupender Singh Sangwan, and Jitendra Singh Jangir. “Art of Hiding: An Introduction to Steganography”. In: ().
- [5] Abbas Cheddad et al. “Digital image steganography: Survey and analysis of current methods”. In: *Signal processing* 90.3 (2010), pp. 727–752.
- [6] Joyshree Nath and Asoke Nath. “Advanced Steganography Algorithm using encrypted secret message”. In: *International journal of advanced computer science and applications* 2.3 (2011).
- [7] P Kaur. “Implementation of Image Processing Algorithms on the Parallel Platform Using Matlab”. In: *International journal of Computer Science and Engineering Technology* 4 (2013), pp. 696–706.
- [8] Ming Qi, Guangzhong Sun, and Guoliang Chen. “Parallel and SIMD Optimization of Image Feature Extraction”. In: *Procedia Computer Science* 4 (2011), pp. 489–498.
- [9] Chenjun Lin and Shangping Zhong. “A Parallel Method for Stego Image Feature Extraction on Multicore CPU”. In: *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2013 5th International Conference*

- [10] Alaa A Jabbar Altaay, Shahrin bin Sahib, et al. "An Introduction to Image Steganography Techniques". In: *Advanced Computer Science Applications and Technologies (ACSAT)*, 2012 International Conference on. IEEE. 2012, pp. 122–126.
- [11] Serdar C,akir and A Enis Cetin. "Mel-cepstral methods for image feature extraction". In: *Image Processing (ICIP)*, 2010 17th IEEE International Conference on. IEEE. 2010, pp. 4577–4580.
- [12] Qiang Hou et al. "Image Feature Extraction Based on Compressive Sensing with Application of Image Denoising". In: *Electrical and Control Engineering (ICECE)*, 2010 International Conference on. IEEE. 2010, pp. 1154–1157.
- [13] Sabahat Saleem et al. "Multi-Core Program Optimization: Parallel Quick Sort in Intel Cilk Plus". In: ().
- [14] Ching-Chi Lin, Pangfeng Liu, and Jan-Jan Wu. "A Novel Approach for Finding Optimization Opportunities in Multicore Architectures". In: *Parallel and Distributed Processing with Applications (ISPA)*, 2011 IEEE 9th International Symposium on. IEEE. 2011, pp. 238–243.
- [15] Arch D Robison. "Composable Parallel Patterns with Intel Cilk Plus". In: *Computing in Science & Engineering* 15.2 (2013), pp. 0066–71.
- [16] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming*, Revised Reprint. Elsevier, 2012.
- [17] Feng-Cheng Huang et al. "High-performance SIFT hardware accelerator for real-time image feature extraction". In: *Circuits and Systems for Video Technology*, IEEE Transactions on 22.3 (2012), pp. 340–351.
- [18] Jatin Chhugani et al. "Efficient implementation of sorting on multi-core SIMD CPU architecture". In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1313–1324.
- [19] Michael Klemm et al. "Extending OpenMP* with vector constructs for modern multicore SIMD architectures". In: *OpenMP in a Heterogeneous World*. Springer, 2012, pp. 59–72.

- [20] Xinmin Tian et al. "Compiling c/c++ simd extensions for function and loop vectorizaion on multicore-simd processors". In: Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International. IEEE. 2012, pp. 2349–2358.
- [21] Y Krishnakumar et al. "Realization of a parallel operating SIMD-MIMD architecture for image processing application". In: Computer, Communication and Electrical Technology (ICCCET), 2011 International Conference on. IEEE. 2011, pp. 98–102.
- [22] Xuanhua Shi et al. "Toward scalable Web systems on multicore clusters: making use of virtual machines". In: The Journal of Supercomputing 61.1 (2012), pp. 27–45.
- [23] Xuanhua Shi et al. "VNIX: managing virtual machines on clusters". In: Frontier of Computer Science and Technology, 2008. FCST'08. Japan-China Joint Workshop on. IEEE. 2008, pp. 155–162.
- [24] Ruijin Xiao et al. "A novel multi-core processor for communication applications". In: Solid-State and Integrated Circuit Technology (ICSICT), 2010 10th IEEE International Conference on. IEEE. 2010, pp. 236–238.
- [25] Stephen W Keckler, Oyekunle Ayinde Olukotun, and H Peter Hofstee. Multicore processors and systems. Springer, 2009.
- [26] Hasitha Muthumala Waidyasooriya et al. "FPGA implementation of heterogeneous multicore platform with SIMD/MIMD custom accelerators". In: Circuits and Systems (ISCAS), 2012 IEEE International Symposium on. IEEE. 2012, pp. 1339–1342.
- [27] Geoffrey Blake, Ronald G Dreslinski, and Trevor Mudge. "A survey of multicore processors". In: Signal Processing Magazine, IEEE 26.6 (2009), pp. 26–37.
- [28] James Donald and Margaret Martonosi. "An efficient, practical parallelization methodology for multicore architecture simulation". In: Computer Architecture Letters 5.2 (2006), pp. 14–14.

- [29] Javier Diaz, Camelia Munoz-Caro, and Alfonso Nino. "A survey of parallel programming models and tools in the multi and many-core era". In: *Parallel and Distributed Systems, IEEE Transactions on* 23.8 (2012), pp. 1369–1386.
- [30] Henry Kasim et al. "Survey on parallel programming model". In: *Network and Parallel Computing*. Springer, 2008, pp. 266–275.
- [31] Horacio González-Vélez and Mario Leyton. "A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers". In: *Software: Practice and Experience* 40.12 (2010), pp. 1135–1160.
- [32] Behrooz Parhami. *Introduction to parallel processing: algorithms and architectures*. Vol. 1. Springer, 1999.
- [33] Mark D Hill and Michael R Marty. "Amdahl's Law in the Multicore Era." In: ().
- [34] Niels Provos and Peter Honeyman. "Hide and seek: An introduction to steganography". In: *Security & Privacy, IEEE* 1.3 (2003), pp. 32–44.
- [35] Shrutika Suri et al. "Comparative Analysis of Steganography for Coloured Images". In: (2014).
- [36] Somdip Dey et al. "Advanced steganography algorithm using randomized intermediate QR host embedded with any encrypted secret message: ASA QR algorithm". In: *International Journal of Modern Education and Computer Science (IJMECS)* 4.6 (2012), p. 59.
- [37] V Lokeswara Reddy, A Subramanyam, and P Chenna Reddy. "Implementation of LSB steganography and its evaluation for various file formats". In: *Int. J. Advanced Networking and Applications* 2.05 (2011), pp. 868–872.

- [38] Gunjan Nehru and Puja Dhar. "A Detailed look of Audio Steganography Techniques using LSB and Genetic Algorithm Approach". In: *International Journal of Computer Science (IJCSI)* 9 (2012), pp. 402–406.
- [39] Atallah M Al-Shatnawi. "A new method in image steganography with improved image quality". In: *Applied Mathematical Sciences* 6.79 (2012), pp. 3907–3915.
- [40] Nitin Jain, Sachin Meshram, and Shikha Dubey. "Image Steganography Using LSB and Edge–Detection Technique". In: *International Journal of Soft Computing and Engineering (IJSCE)* ISSN 223 (2012).
- [41] Tomáš Pevný, Tomáš Filler, and Patrick Bas. "Using high-dimensional image models to perform highly undetectable steganography". In: *Information hiding*. Springer. 2010, pp. 161–177.
- [42] Jicang Lu et al. "Steganalysis of LSB Matching Based on Image Noise". In: *Multimedia and Ubiquitous Engineering*. Springer, 2014, pp. 211–217.
- [43] Zhang, Li Zhang, and Li. "A novel blind detector for additive noise steganography in JPEG decompressed images". In: *Multimedia Tools and Applications*. Springer, 2014.
- [44] Dr Ekta Walia, Payal Jain, and Navdeep Navdeep. "An analysis of LSB & DCT based steganography". In: *Global Journal of Computer Science and Technology* 10.1 (2010).
- [45] Shuting Xu and Shuhua Lai. "An Optimal Least Significant Bit Based Image Steganography Algorithm". In: *Proceedings of International Conference on Internet Multimedia Computing and Service*. ACM. 2014, p. 42.
- [46] Manu Devi and Nidhi Sharma. "Improved detection of least Significant bit steganography algorithms in color and gray scale images". In: *Engineering and Computational Sciences (RAECS)*, 2014 Recent Advances in. IEEE. 2014, pp. 1–5.

- [47] Mingqiang Yang, Kidiyo Kpalma, Joseph Ronsin, et al. “A survey of shape feature extraction techniques”. In: Pattern recognition (2008), pp. 43–90.
- [48] Hao Feng et al. “Parallelization and characterization of SIFT on multi-core systems”. In: Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on. Sept. 2008, pp. 14–23. doi: 10.1109/IISWC.2008.4636087.
- [49] Qi Zhang et al. “SIFT implementation and optimization for multi-core systems”. In: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on. Apr. 2008, pp. 1–8. doi: 10.1109/IPDPS.2008.4536131.
- [50] In Kyu Park et al. “Design and Performance Evaluation of Image Processing Algorithms on GPUs”. In: Parallel and Distributed Systems, IEEE Transactions on 22.1 (Jan. 2011), pp. 91–104. issn: 1045-9219. doi: 10.1109/TPDS.2010.115.
- [51] J. Clemons et al. “EFFEX: An embedded processor for computer vision based feature extraction”. In: Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE. June 2011, pp. 1020–1025.