

MULTICORE SPECIFICATION GENERATION SYSTEM

BY

NNADI HILLARY SUNDAY

PG/MSC/12/62672

BEING A M.SC. PROJECT SUBMITTED IN PARTIAL
FULFILMENT FOR THE AWARD OF MASTER OF
SCIENCE DEGREE IN COMPUTER SCIENCE OF
UNIVERSITY OF NIGERIA, NSUKKA

SUPERVISOR: DR. M.C. OKORONKWO

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF NIGERIA, NSUKKA

NOVEMBER, 2014

TITLE PAGE

MULTICORE SPECIFICATION GENERATION SYSTEM

APPROVAL PAGE

This project report is approved for submission.

Dr M.C OKORONKWO

Supervisor

CERTIFICATION PAGE

I hereby declare that the work presented herein was done by me, and not by third party.
Should I be convicted of having cheated in this work, I shall accept the verdict of the
university.

NNADI HILLARY SUNDAY

PG/MSC/12/62672

DEDICATION

I dedicate this work to God almighty and to those who believed in me.

ACKNOWLEDGMENTS

I am most indebted to my great supervisor Dr. M.C. Okoronkwo for his fatherly contribution and thorough supervision of my work. I am so grateful to him for the knowledge he impacted on me during my course work. He has been a wonderful teacher/lecturer in the department.

I am so grateful to my brothers, sisters and friends who have being there for me. I thank them all for supporting me throughout my time in the university. They have made significant contribution for me both financially and materially.

I acknowledge the efforts of Dr. Atabong, Prof. Bakpo F.S, Dr. Ebem and Sultan who helped me in so many ways to see the success of this work.

Finally, special thanks to God almighty who has being keeping me alive till this moment I say may your name be glorified.

ABSTRACT

Performance analysis is the task of monitoring the behaviour of a program execution. The main goal is to find out the possible adjustments that might be done in order to improve the performance of the computer system in use. To be able to get that improvement, it is necessary to find the different causes/contributors of overhead. Today, we are already in the multicore era, but there is a gap between the level of development of the two main divisions of multicore technology (hardware and software).

This project is focused on the issues concerning performance analysis, tuning of applications running specifically in a shared memory system and development of application that automatically extract system characteristics and configurations. This application is developed using OODM and implemented using C# programming language and can be used on any windows Operating System. The application developed from this project critically analyses multicore system, determine various causes of overhead in multicore environment, extracts system parameters and present various optimization strategies.

TABLE OF CONTENTS

Title Pagei
Approval Pageii
Certification Pageiii
Dedicationiv
Acknowledgmentsv
Abstractvi
Table of Contentsvii
List of Figuresx
List of tablesxi
Chapter 1: Introduction												
1.1	Introduction1
1.2	Statement of Problem4
1.3	Objective of the Study5
1.4	Significance of Study5
1.5	Scope of Study6
1.6	Definition of Terms6
Chapter 2: Literature Review												
2.0	Introduction7
2.1	Theoretical Background7
2.2	Review of Related Literature10
2.2.1	Multicore Performance Analysis11
2.2.2	Multicore CPU performance evaluation11
2.2.3	Factors Affects the Performance12
2.2.4	Multicore CPU Benchmarking13
2.2.5	Multicore CPU power benchmarks14
2.2.6	Example of Multicore CPU Performance Analysis14
2.2.7	Heterogeneity Vs Homogeneity16
2.2.8	Memory Hierarchy and interconnection17
2.2.9	Multicores Optimization20
Chapter 3: System Analysis and Design												
3.0	Introduction22
3.1	Description of Existing System23

Appendix F: HardwareInfo class source code.....63
Appendix G: Hard Drive control source code.....72
Appendix H: Motherboard Control source code74
Appendix I: Network Control source code76
Appendix J: Optical drives control source code78
Appendix K: Operating system control source code79
Appendix L: Performance control source code81
Appendix M: Peripherals control source code84
Appendix N: RAM control source code86
Appendix O: Summary control source code87

LIST OF FIGURES

Fig. 1.1. An example of a dual core MCP (Multicore Processor) structure.....4
Fig. 2.1 shows the result of the servers based on Intel multicore CPU run times15
Fig. 2.2 shows the increasing of power consumption due to the number of VMs16
Fig. 2.3. A basic homogenous Chip Multiprocessor17
Fig. 2.4. An example of a heterogeneous Chip Multiprocessor17
Fig. 2.5 Memory access hierarchy organization of general processors units18
Fig. 2.6-7. Estimation of speed accesses scale in a processor unit19
Fig. 3.1: AutoSpec Use Case Diagram26
Fig. 3.2: Class diagram27
Fig. 3.3: Save Specification as text file.....33
Fig. 4.1: System Block Diagram33
Fig. 4.2: System flowchart34
Fig. 4.3: Auto Spec summary screenshot37
Fig. 4.4: Operating System details screenshot38
Fig. 4.5: CPU details screenshot39
Fig. 4.6: RAM details screenshot39
Fig. 4.7: Motherboard details screenshot40
Fig. 4.8: Graphics details screen41
Fig. 4.9: Hard Drives details screenshot41
Fig. 4.10: Optical Drives details screenshot42
Fig. 4.11: Audio details screenshot43
Fig. 4.12: Peripheral details screenshot43
Fig. 4.13: Network details screenshot.....44
Fig. 4.14: Performance details screenshot44

LIST OF TABLES

Table 4.1: List of output screen controls35
---	-------	-------	-------	-------	-------	-------	------

CHAPTER ONE

INTRODUCTION

1.1 Introduction

With computers playing an increasingly critical role in our day-to-day lives, it is important to know their components and how each works and of what impact they impose on performance of the computer system.

According to (Arnold, 1994) Computer performance is characterised by the amount of useful work accomplished by a computer system compared to time and resources used. Depending on the context, good computer performance is dependent on the available system resources. Most computer users do not know the system specification, they lack the knowledge of conventional way of extracting system parameters but with the computerised system in this thesis (Otherwise known as Autospec) every computer users will be able to determine the system configuration by installing and running the software.

The System development can be likened to building a house, this demands adequate planning and preparation in order to meet the objectives of the proposed design.

The parameters or the resources that are of interest in our analysis include the followings:

- Summary
- Operating system
- CPU
- RAM
- Hard drives
- Optical drives
- Motherboard
- Graphics
- Network
- Audio
- Peripheral
- Performance

Performance analysis is the task of investigating the behaviour of program execution (Mario, 2009). The main aim is to find out the possible adjustments that might be done in order enhance the performance of computer system. Besides, the hardware architecture and software platform (operating system) where a program is executed has impact on its performance. Workload characterization involves studying the user and machine environment, observing key characteristics, and developing a workload model that can be used repeatedly. Once a workload model is available, the effect of changes in the workload and system can be easily evaluated by changing the parameters of the model. This can be achieved by using compiler directives such OpenMP multithread application. In addition, workload characterization can help you to determine what's normal, prepare a baseline for historical comparison, comply with management reporting, and identify candidates for optimization.

Presently, multicore processors chips are being introduced in almost all the areas where a computer is needed. For example, many laptop computers have a dual core processor inside. High Performance Computing (HPC) address different issues, one of them is the exploitation of the capacities of multicore architecture(Mario, 2009).

Presently, multicore processors chips are being introduced in almost all the areas where a computer is needed. For example, many laptop computers have a dual core processor inside. High Performance Computing (HPC) address different issues, one of them is the exploitation of the capacities of multicore architecture.

Performance analysis and optimization is a field of HPC responsible for analysing the behaviour of applications that perform big amount of computation. Some applications that perform high volume of computations require analysing and tuning. Therefore, in order to achieve better performances it is necessary to find the different causes of overhead.

There are a considerable number of studies related to the performance analysis and tuning of applications for supercomputing, but there are relatively few studies addressed specifically to applications running on a multicore environment.

A multicore system is composed of two or more independent cores (or CPUs). The cores are typically integrated onto a single circuit die (known as a chip multiprocessor or CMP), or they may be integrated onto multiple dies in a single chip package.

This thesis examines the issues involved in the performance analysis and tuning of applications running specifically in a shared Memory and the development of a computerized system for retrieving systems specification for possible changes. Multicore hardware is relatively more mature than multicore software, from that reality arises the necessity of this research. We would like to emphasize that this is an active area of research, and there are only some early results in the academic and industrial worlds in terms of established standards and technology, but much more will evolve in the years to come.

Several years, the computer technology has been going through a phase of many developments. Based on Moore law, the speed of processors has been increasing very fast. Every new generation of micro-processor comes with clock rate usually twice or even much faster than the previous one. That increase in clock frequency drove increases in the processors performance, but at the same time, the difference between the processors speed and memory speed was increasing. Such gap was temporarily solved by instruction level parallelism (ILP) (Faxen et al, 2008). Exploiting ILP means executing instructions that occur close to each other in the stream of instructions through the processor in parallel. Though it appeared very soon that more and more cycles are being spent not in the processor core execution, but in the memory subsystem which includes the multilevel caching structure, and the so-called Memory Wall, problem started to evolve quite significantly due to the fact that the increase in memory speed didn't match that of processor cores.

Very soon a new direction for increasing the overall performance of computer systems had been proposed, namely changing the structure of the processor subsystem to utilize several processor cores on a single chip. These new computer architectures received the name of Chip Multi Processors (CMP) and provided increased performance for new generation of systems, while keeping the clock rate of individual processors cores at a reasonable level. The result of this architectural change is that it became possible to provide further improvements in performance while keeping the power consumption of the processor subsystem almost constant, the trend which appears essential not only to power sensitive market segments such as embedded systems, but also to computing server farms which suffer power consumption/dissipation problems as well.

Some of the advantages that shared memory CMP may offer are:

- Direct access to data through shared memory address space.
- Greater latency hiding mechanism.
- MP's appears to lower power and cooling requirements per FLOP.

There are two main types of CMP

- There are those that contain a few very powerful cores, essentially the same core one would put in a single core processor. Examples include AMD Athlons, Intel Core 2, IBM Power 6 and so on.
- There are those systems that trade single core performance for number of cores, limiting core area and power. Examples include the Tiler 64, the Intel Larrabee and the Sun UltraSPARC T1 and T2 (also known as Niagara 1-2). Figure 1.1 shows the basic structure of a dual core processor

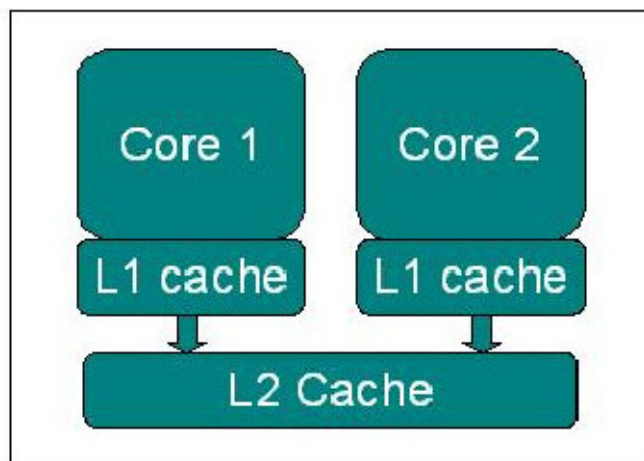


Fig. 1.1. An example of a dual core MCP (Multicore Processor) structure.

1.2 Statement of Problem

Multicore hardware technology is advancing very fast. There is a gap between the level of development of the multicore hardware and multicore software technology. Multithread applications may not always take proper advantage of multicore hardware architecture, especially when a user of such system lack the in depth knowledge of the system parameters.

Most people can name processor, maybe how much RAM it has and how big the hard drive is through the stickers or tags on the PC and a user can also get these basic information by right clicking my computer and then clicking on properties but these miss out lots of information that you need to know about your system such as graphic card, motherboard type, cache type and temperature of some hardware etc

1.3 Objective of the Study

The main objectives of this work are:

- ✓ To develop an application that would help all the computer user to automatically extract system parameters
- ✓ Determine the main causes of overhead on the multicore environment.
- ✓ To apply the necessary measures/changes to make a better use of the hardware resources and therefore obtain a better performance.

1.4 Significance of Study

Autospec is a software that acts as a sticker for your PC. It analyses and shows the statistics on every piece of hardware in your computer. Including CPU, Motherboard, RAM, Graphics card, Operating systems, Hard Disks, Optical Drives, Audio support. Additionally Autospec adds the temperatures of your different components, so you can easily see if there is a problem and also an interface for checking for software updates.

It certainly helps any PC user especially novices in everyday computing life to Troubleshoot, diagnosis, make replacement, control process and implement some settings etc.

1.5 Scope of Study

The multicore technology may be divided into two main categories, hardware and software. But the application focuses on the hardware. The developed software will be able to extract the systems specification in all multicore computer system running on windows Operating System.

1.6 Definition of Terms

MULTICORE: Multicore refers to an architecture in which a single physical processor incorporates the core logic of more than one processor. A single integrated circuit is used to package or hold these processors. These single integrated circuits are known as a die.

MULTITHREADING: is a type of execution model that allows multiple threads to exist within the context of a process such that they execute independently but share their process resources. A thread maintains a list of information relevant to its execution, including the priority schedule, exception handlers, a set of CPU registers, and stack state in the address space of its hosting process.

TUNNING: is the improvement of system performance by varying parameters such as Processor, secondary storage devices, main memory and application modification.

APPLICATION PATTERN: This is a design pattern which covers useful architectural and code design patterns that can lead to the creation of more maintainable and evolvable software.

OpenMP (Open Multi-Processing): is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most processor architectures and operating systems, including Solaris, AIX, HP-UX, GNU/Linux, Mac OS X, and Windows.

COMPUTER PERFORMANCE: is characterized by the amount of useful work accomplished by a computer system compared to the time and resources used

CHAPTER TWO

LITERATURE REVIEW

2.0 Introduction

Multicore is the new trend in computer world. As technology advances more cores are built to enhance system speed, functionality, efficiency and dependency. In the era of single processor, most computers were slow and there was a heavy limitation to file processing and other data manipulations. Dual core systems also produced undesirable power consumption and as result alternative research has encouraged the production of multicore systems. System specification is an area in which some average computer literates find difficult to handle. Many people don't know how to check their system configuration. This project presents a platform that will allow a novice to perform such activity.

The various system specification categories have been listed earlier and we are more interested on the technology used in developing them in this section.

2.1 Theoretical Background




AutoSpec is a software package developed in this project work to help analyse the configuration of a computer system. The software makes life easier for users who may be clamouring to know the stuff their system units are made up of. It takes above average computer literate to locate where most of computer specification are hidden.

C# programming language was used to build this application. The system is actually a desktop application meaning that it can be installed on any PC and ran as a standalone software. It is not an internet based software in that no internet connection is needed to use it. The technology used in this project work is based on system **management** feature of Microsoft .Net framework. This program makes use of Windows Management Instrumentation (WMI) infrastructure to explore the system properties. In .Net framework System.Management namespace handle anything concerning system properties. Management Namespace provides access to a rich set of management information and management events about the system, devices, and applications instrumented to the Windows Management Instrumentation (WMI) infrastructure. Applications and services can query for management information (such as how much free space is left on the disk, what is the current CPU utilization, which database a certain application is connected to,








and much more), using classes derived from `ManagementObjectSearcher` and `ManagementQuery`, or subscribe to a variety of management events using the `ManagementEventWatcher` class. The accessible data can be from both managed and unmanaged components in the distributed environment.

One powerful class in the `System.Management` namespace used in this project work is **`ManagementObjectSearcher`**. It retrieves a collection of management objects based on a specified query. This class is one of the more commonly used entry points to retrieving management information. For example, it can be used to enumerate all disk drives, network adapters, processes and many more management objects on a system, or to query for all network connections that are up, services that are paused, and so on. When instantiated, an instance of this class takes as input a WMI query represented in an `ObjectQuery` or its derivatives, and optionally a `ManagementScope` representing the WMI namespace to execute the query in. It can also take additional advanced options in an `EnumerationOptions`. When the `Get ()` method on this object is invoked, the `ManagementObjectSearcher` executes the given query in the specified scope and returns a collection of management objects that match the query in a `ManagementObjectCollection`. The `ManagementObjectSearcher` type exposes the following members.

Constructors





	N a m e	D e s c r i p t i o n
	<code>ManagementObjectSearcher ()</code>	Initializes a new instance of the <code>ManagementObjectSearcher</code> class.
	<code>ManagementObjectSearcher(ObjectQuery)</code>	Initializes a new instance of the <code>ManagementObjectSearcher</code> class used
	<code>ManagementObjectSearcher(String)</code>	Initializes a new instance of the <code>ManagementObjectSearcher</code> class

Methods

	N a m e	D e s c r i p t i o n
	CreateObjRef	Creates an object that contains all the relevant information required to generate
	Dispose ()	Releases all resources used by the Component. (Inherited from Component.)
	Dispose(Boolean)	Releases the unmanaged resources used by the Component and optionally releases the managed resources. (Inherited from Component.)
	Equals(Object)	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Releases unmanaged resources and performs other cleanup
	Get ()	Invokes the specified WMI query and returns the resulting collection.
	Get(ManagementOperationObserver)	Invokes the WMI query asynchronously, and binds to a watcher to deliver the results.

Another important class used most times is **ManagementClass** class. It represents a Common Information Model (CIM) management class. A management class is a WMI class such as Win32_LogicalDisk, which can represent a disk drive, and Win32_Process, which represents a process such as Notepad.exe. The members of this class enable you to access WMI data using a specific WMI class path.

Constructors

	N a m e	D e s c r i p t i o n
	ManagementClass ()	Initializes a new instance of the ManagementClass class. This is the default constructor.
	ManagementClass(ManagementPath)	Initializes a new instance of the ManagementClass class. The class represents a Common Information.
	ManagementClass(String)	Initializes a new instance of the ManagementClass class initialized
	ManagementClass(ManagementPath, ObjectGetOptions)	Initializes a new instance of the ManagementClass

2.2 Review of Related Literature

2.2.1 Multicore Performance Analysis

In computer and semi-conductors industry Multicore CPU has become the standard for the current era of processors development due to the significant level of performance it offers. With the variety of multiple multicore architecture and different levels of performance, it becomes imperative to compare the architectures of multicore to ensure that the performance aligns itself with the expected specifications.

Moore's law of in-chip performance doubling is a standard for development in the computer and semiconductor industry. Memory capacity and CPU speeds are two of the many digital electronic devices believed by Moore's law to gain increased development approximately every 18 months. Multicore CPUs have evolved in this process and have become a vital part of our daily life. They are implemented in many devices, and their performance varies considerably by application. This situation draws the attention of researchers to evaluate CPU performance carefully to obtain higher performance at lower cost.

The high performance speed gained by multi-processors (dual core), produce uncontrollable high power consumption, based on that, alternative research trends encouraged the production of multicore CPUs in order to minimize power consumption, while simultaneously increasing the high processing speed. The architecture of multicore CPUs is responsible for high speed achieved by hungry applications with lower power consumption, each task is shared among the cores.

Performance analysis is a criterion that defines the performance of a system, and it is necessary at every stage of the computer system life-cycle, to ensure high performance at a given cost. The need for performance analysis was derived by radical changes in a number of factors including;

i. Current computer user who is more demanding than computer users years ago.

ii. The popularity of computer technology, which is no longer a hidden fact, necessitated inundation in the computer market of different computer manufactures, each differing in performance. To be able to achieve improvement in multicore it is also necessary to find the different causes of overhead(Mario, 2009). Such changes require performance analysis that meet users need and help select a better alternative which provides higher performance at given cost implementing trade-offs between what each technique provides and the required criteria in mind.

In review of related work we explored different ways used to evaluate multicore performance, and the proper method for selecting a reliable evaluation techniques, metrics, and measure of multicore CPUs. We looked at the techniques used to evaluate multicore CPUs and the considerations that should observed when selecting the techniques to be used.

2.2.2 Multicore CPU performance evaluation

To evaluate the performance of multicore, we have to adopt a good evaluation technique; the following are different techniques in use

- Analytical modelling
- Simulation
- Measurement.

Certain criterions are to be observed during multicore performance evaluation. To prove the correctness of technique used we require the use of second evaluation technique. For instance, an evaluation done with simulation should be validated using the second technique such as analytical or measurement. The considerations for selecting the appropriate technique is listed below as provided by (Jain, 1991).

- ✓ Stage of evaluation
- ✓ Time requirement
- ✓ Tools required
- ✓ Accuracy of result

- ✓ Cost of technique in use
- ✓ Saleability of the product

He stated that analytical modelling can be performed at any stage of the system life-cycle as it requires less amounts of time than simulation and measurement because they both vary with evaluation time, analytical modelling needs no tools for analysis, but it may give less accurate results. Analytical cost less in terms of capital compared to the other techniques. Unfortunately the saleability for such products with just analytical modelling performance result is low.

Other Metrics that are used in evaluating multicore CPUs performance

According to various authors as stated below, the following metrics are used;

- ✓ Memory bandwidth: the rate of data sustained from the CPU core to the RAM (Random Access Memory) (Kayi et al, 2007) the CPU has a direct access to RAM.

- ✓ Throughput: this is the average rate of successful processes (Sharma, et al 2009).

According to (Monchiero et al, 2006).

- ✓ Execution time: The time needed to complete program execution, it could be referred to as the processing time.

- ✓ Energy: The power needed to run a program

- ✓ Memory latency: this is the time delay between the memory controller signalled the memory module to access data from the RAM and the time the data become available for the memory module, also known as CAS (Column Address Strobe) latency.

- ✓ Percentage of memory contention: the percentage among the cores trying to access the RAM at the same time.

- ✓ Response time: The time that the user finishes the request and the time the system starts a response.

2.2.3 Factors Affects the Performance

Factors are the performance parameters that we want to study to see their effects on the system. Factors also depend upon the required performance needed to utilize the CPU and get the expected outcome from it. In this subsection we are going to show examples of factors that affect multicore CPU performance.

- ❖ Memory: Memory architecture used and memory speed, can affect the performance of the multicore CPU (Sharma et al,2009).
- ❖ Scalability: Affects performance based on the rate of increase of the workloads (i.e. tasks) (Carpenter, 2007).
- ❖ I/O bandwidth: Can affect the performance by utilizing the CPU cores which leads to more resources consumption without performance increasing (Sharma et al,2009).
- ❖ Inter-core communication: The interaction between cores in multicore CPU's can be implemented by various mechanisms, affecting overall CPU performance due to shared workloads between cores (Sharma et al,2009).
- ❖ Operating system (OS): OS is the manager for the CPU and it assigned tasks to cores based on a scheduling mechanism, affecting the multicore CPU performance (Pase and Eckl 2005).
- ❖ CPU clock speed: Clock speed has an impact on processor performance with slow clock speed reducing throughput (Pase and Eckl 2005).
- ❖ Numbers of cores: this affects the CPU performance as multicore architecture workload is divided between the cores (Pase and Eckl 2005).
- ❖ Cache coherent: Multicore architectures uses different caching mechanisms as the cache is shared among the cores, causing cache coherent to affect CPU performance. (Kayi et al, 2007).

These are some examples of the factors affecting multicore CPU performance, and for the analysis of each factor under study we will define ways to optimize the performance by analyzing the effects of the factors and interpret the result to get the optimal expected performance.

2.2.4 Multicore CPU Benchmarking

Multicore CPUs are designed for a variety of applications, (virtualization, Games, and Embedded systems), with this kind of diversity, measuring the performance for multicore systems became a necessity to ensure that the performance delivered are as required by the system. Different tools are used to measure multicore CPU performance. Tools like profiling, which is used to monitor and observe system performance behaviour rather than measuring. Measuring elapse time for the processes in multicore architectures having multiple threads,

results in high-level information of processing speeds which increases performance as result of parallelization (Prinslow, 2011). Utilizing benchmark tools often results in better measurements that become more relevant and accurate to system profiling. Below are the benchmarking approaches used to measure multicore CPU performance.

2.2.5 Multicore CPU power benchmarks

Following the evolution of mobile devices and computers, power efficiency has become an important aspect that requires new approaches to measure multicore CPU performance. Benchmark companies developed new benchmarks depending on power to measure the performance. The two power benchmarks considered as the industry standard power benchmarks (Domeika, 2009).

- ❖ EEMBC EnergyBench: Uses average power (in watts) and energy (in Joules per iteration) as a performance metric to measure the performance. Initially it runs a multicore EEMBC benchmark and provides power measurement simultaneously, measuring performance and power by monitoring the power rails on system board at execution time.
- ❖ BDTI Benchmark suite: Used to estimate or measure processor power efficiency. In estimating power, it uses consistent assumptions and conditions to give accurate estimations that are helpful in making decisions. In measuring performance it uses vendor benchmarks for the specific targets of measurement.

Benchmark results for multicore CPU performance depend on the test run by the benchmark to measure the performance of the multicore CPU for specific applications, and by defining the reasoning for measurements we can relate different multicore CPU performances to each other based on the benchmark used to utilize and measure the multicore CPU performance.

2.2.6 Example of Multicore CPU Performance Analysis

The following are some example of performance analysis processes for multicore CPUs that will assist in selecting the proper CPU for a machine specification.

Server virtualization of Multicore CPU: Intel IT (Information Technology) team evaluated server performances based on three Intel multicore CPU servers (A Four-socket server based on Quad-

Core Intel Xeon CPU X7350 with 16 cores, a dual-socket server based on Quad-Core Intel Xeon CPU X5355 with 8 cores and a dual-socket server based on Intel Dual-Core Xeon CPU 5160 with 4 cores) (Carpenter, 2007).

In comparing the performance of the multicore CPUs, the Intel IT team targeted the speed of the CPUs and the power efficiency. Due to CPU clock speed, runtime used to measure the performance on each CPU, the data was normalized. Furthermore, the normalized workload consists of VMs (Virtual Machines) and a copy of a synthetic CPU intensive DB application in each VM.

W-M/Job (Watt-minute per job) metrics was utilized to measure CPU power efficiency with an increasing workload to test the scalability factors of the CPUs (Carpenter, 2007). The results from the three servers show different levels of scalability in terms of power consumption. As the VMs number increased the run time remains constant until the workload equals the number of cores. After the number of VMs exceeds the number of cores, the run time begins to increase,

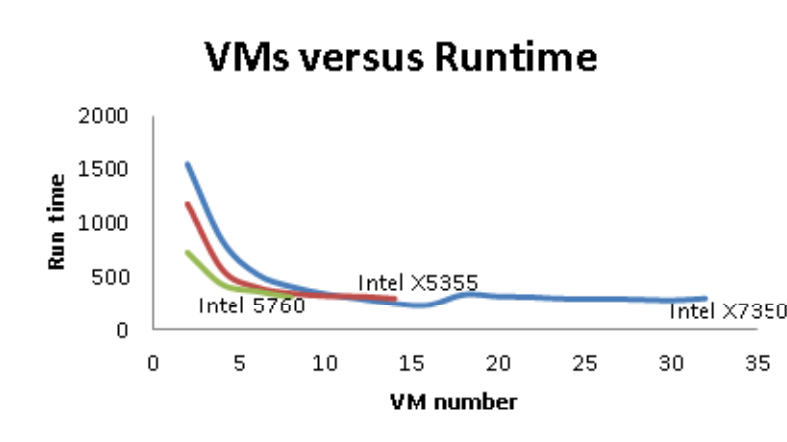


Fig. 2.1 shows the result of the servers based on Intel multicore CPU run times(Jain, 1991).

We can observe that the Intel X7350 CPU run time is almost constant, until the number of VM reach 16, then it starts to increase. Alternatively, in the case of the Intel X5355, the run time starts to increase around 6 VMs, and around 4 VMs were running the Intel 5760.

Another approach that is useful in performance evaluation of the CPUs is to measure the power consumption of the different CPUs based upon increasing workload to test the scalability (Carpenter, 2007). The result of the test shows that Quad-core Intel Xeon CPU X7350 based

servers, consumed more power than its alternatives due to the larger number of cores. Consuming at an average of 495 W (Watts) on 2 VMs running, 478 W for Quad-core Intel Xeon CPU X5355, and average of 330 W for Dual-core Intel Xeon CPU 5160. As the number of VMs increased, the servers became more power efficient which can be observed from figure 2. The Quad-core Intel Xeon CPU X7350 based servers with the maximum workload showed the power consumption per job decrease from the start and that is due to scalability of the CPU.

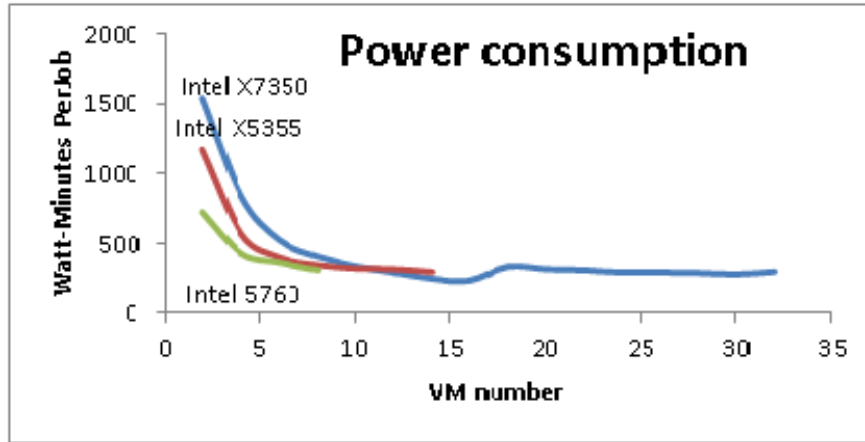


Fig. 2.2 shows the increasing of power consumption due to the number of VMs Jain, 1991).

This section reviewed examples of how performance analysis works, by comparing the multicore CPUs. We can use the result to make proper decisions in terms of selecting the appropriate CPU for a required performance level.

2.2.7 Heterogeneity Vs Homogeneity

In a multicore chip, the cores could be identical or there could be more than one kind of core. Our research is focused on a homogeneous multicore environment. But we consider important to explain the differences between homogeneous and heterogeneous CMP. There are two levels of heterogeneity depending on whether the cores have the same instruction set or not.

Hence there are three possibilities:

- Identical cores, as in most current multicore chips from the Intel Core 2 to the Tiler 64.
- Cores implementing the same instruction set but with different non-functional characteristics.

- Cores with different instruction sets like in the Cell processor where one core implements the PowerPC architecture and 6-8 synergistic processing elements implement a different RISC instruction set.

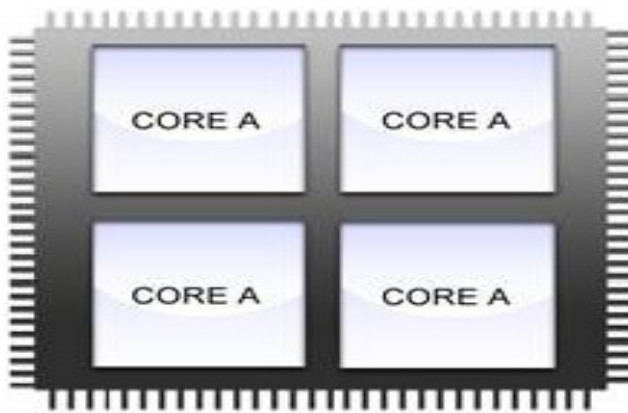


Fig. 2.3. A basic homogenous Chip Multiprocessor(Faxen, 2008).

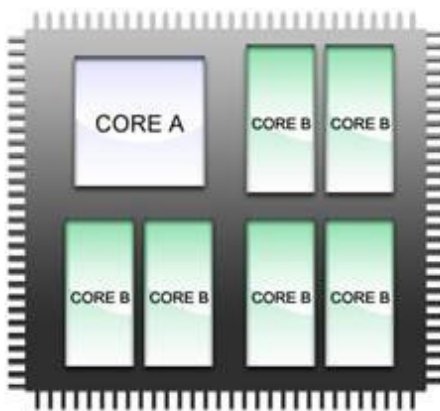


Fig. 2.4. An example of a heterogeneous Chip Multiprocessor(Faxen, 2008).

2.2.8Memory Hierarchy and interconnection

One of the major challenges facing computer architects today is the growing discrepancy in processor and memory speed (Chapman et al, 2008). Processors have been consistently getting faster. But the more rapidly they can perform instructions, the quicker they need to receive the values of operands from memory. Unfortunately, the speed with which data can be read from and written to memory has not increased at the same rate. Memory access time is increasingly the

bottleneck in overall application performance. As a result, an application might spend a considerable amount of time waiting for data. This not only negatively impacts the overall performance, but the application cannot benefit much from a processor clock-speed upgrade either.

In response, the vendors have built computers with hierarchical memory systems, in which a small, expensive, and very fast memory called cache memory (or “cache” for short), supplies the processor with data and instructions at high rates. Each processor of a shared memory system needs its own private cache if it is to be fed quickly; hence, not all memory is shared. Most programs have a high degree of locality in their accesses. Memory hierarchy tries to exploit locality.

There two types of locality:

- Spatial locality: When accessing things nearby previous accesses
- Temporal locality: When reusing an item that was previously accessed.

Figures below show an example the organization of processors memory hierarchy.

Figure 5 shows the organization with two levels of cache memory. Nowadays an additional level (level 3) is used in many multicore processors.

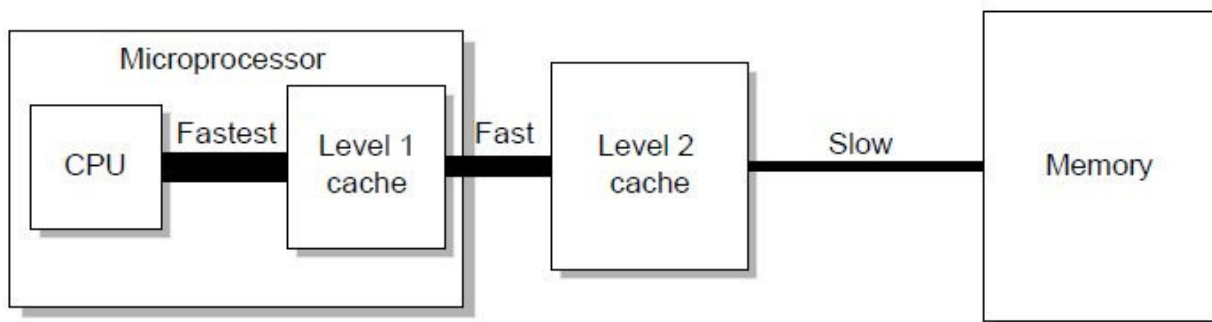


Fig. 2.5 Memory access hierarchy organization of general processors units.

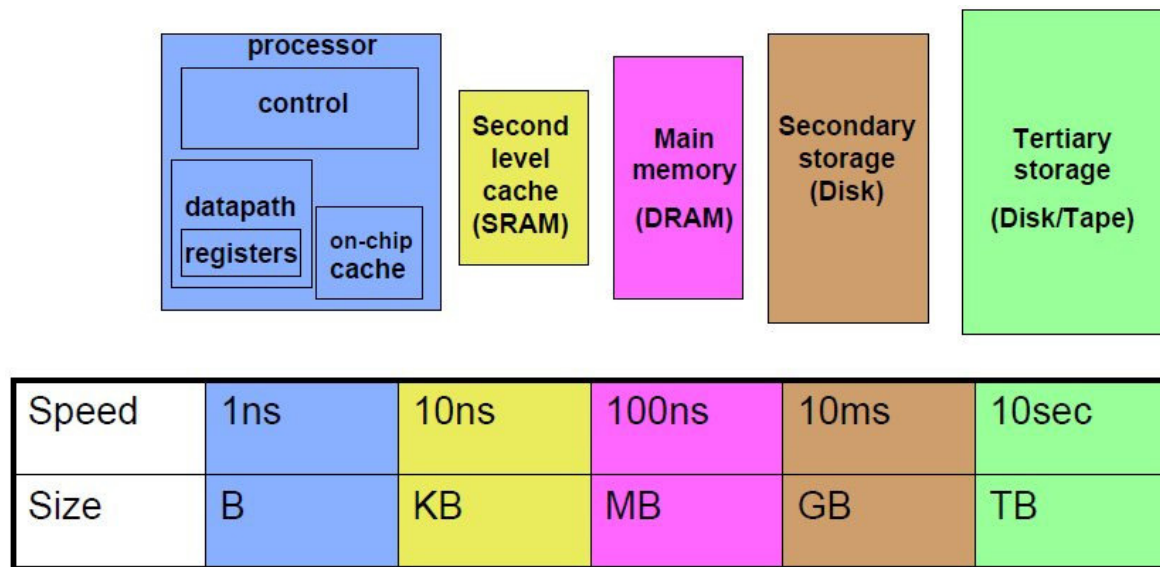


Fig. 2.6-7. Estimation of speed accesses scale in a processor unit

One salient characteristic of multicore architectures is that they have a varying degree of sharing of caches at different levels. Most of the architectures have cores with a private L1 cache. Depending on the architecture, an L2 cache is shared by two or more cores; and an L3 cache is shared by four or more cores. The main memory is typically shared by all cores. The degree of sharing at a level varies from one multicore processor to another. When we speak of CMP we are speaking of shared memory at some levels. Most multicore designs provide some form of coherent caches that are transparent to software. First level caches are typically private to each core and split into instruction and data caches, as in the preceding generation of single core processors (Faxen et al, 2008).

- Early dual core processors had private per core second level caches and were essentially a double single core processor with a minimum of glue logic and an essentially snooping coherence protocol. Some designs continue with separate L2 caches, like the Tiler 64 where each core has a 64 KB L2 cache. However, the glue logic is in this case anything but simple and amounts to a directory based cache coherency protocol on a mesh interconnect.

Second level caches can be shared between the cores on a chip; this is the choice in the Sun Niagara (a 3MB L2 cache) as well as the Intel Core 2 Duo (typically 2-6MB)(Faxen et al, 2008).

- Separate L2 caches backed by a shared L3 cache as in the AMD Phenom processor (512 KB L2 per core, shared 2MB L3) or the recent Intel Core i7 (256 KB L2 per core, shared 8MB L3).

- A hierarchy where L2 caches are shared by subsets of cores. This pertains to the four core Intel Core 2 Quad, which is essentially two Core 2 Duo in a single package. Each of the chips have an L2 cache shared between its two cores but the chips have separate caches. With private L2 caches, the L1-L2 communication is local, and the intercore interconnect is located below the L2 cache, whereas with a shared L2 it sits between the L1 and L2 caches. In the shared case, all L1 misses go over the interconnect whereas in the private case only those that also miss in the L2 do so. This requires a more expensive, low latency interconnect (often a crossbar) which uses a lot of area that could otherwise be used for larger caches (or more cores). Also, L2 access time is increased by the need to go over the interconnect (Faxenet al, 2008). On the other hand, private L2 caches might waste chip area by having the same data occupy space in several caches, and accessing data in the L2 of another core, something that is sometimes needed to achieve cache coherency, becomes more expensive than accessing a shared L2 cache.

2.2.9 Performance analysis and Tuning

This research also aims to provide a review of guide to good practices when analysing and tuning an OpenMP parallel application. Also we provide an explanation of the tools used to analyse the execution performance of those applications. This research provides a set of key factor to have in mind when parallelizing a multithread application. In order to find the reasons of a poor performance, it is necessary to understand how the application is organized. As we said in the introduction to this thesis, we haven't followed the general model where the analysis of the application is done taking the application as a black box. Our work differs from the general model in that, doing the analysis on that way might be sometimes more difficult due to the lack of knowledge of the application's task flow. Therefore we have gone one step ahead in the analysis of the applications through extraction of the applications structure and execution patterns.

(West, 2008) stated the parameters that must be considered when we analyse the performance and tuning of multithread applications.

The key attributes that affect parallel performance are:

- ✓ Coverage
- ✓ Granularity
- ✓ load balancing
- ✓ locality and
- ✓ Synchronization.

The first three are fundamental to parallel programming on any type of machine. However locality is a very important issued to have in mind when optimizing a multithread application in a shared memory system. Their effects are often more surprising and harder to understand, and their impact can be huge.

Chapter 3: System Analysis and Design

3.0 Introduction

With computers playing an increasingly critical role in our day-to-day lives, it is important to know their components and how each works and of what impact they impose on performance of the computer system.

Computer performance is characterised by the amount of useful work accomplished by a computer system compared to time and resources used. Depending on the context, good computer performance is dependent on the available system resources. Most computer users do not know the system specification, they lack the knowledge of conventional way of extracting system parameters. This is the core issue addressed in this thesis (Otherwise known as Autospec) every computer users will be able to determine the system configuration by installing and running the software.

The System development can be likened to building a house, this demands adequate planning and preparation in order to meet the objectives of the proposed design.

The parameters or the resources that are of interest in our analysis include the followings:

- Summary
- Operating system
- CPU
- RAM
- Hard drive
- Optical drives
- Motherboard
- Graphics
- Network
- Audio
- Peripheral
- Performance

3.1 Description of Existing System

What's in your computer? Most people can probably name the processor (Intel or AMD, Celeron or Pentium), maybe how much RAM it has, and maybe how big the hard drive is.

The computer stores display many bright shiny PCs laid out next to each other, most will have tags or stickers indicating the:

- ✓ Processor brand and model
- ✓ Hard drive size and speed
- ✓ Amount of memory (RAM)
- ✓ Graphics card
- ✓ Operating system etc

Some years later, when you need to upgrade your computer, that tag or sticker may be long gone.

The problem with the current/existing system is that. Some of the basic information can be found by right-clicking my Computer and then clicking Properties. The General tab lists some statistics, and the Device Manager on the Hardware tab lists all of the hardware you have installed. But it misses out lots of information that you need and much time is spent hunting around window sourcing for these information. Such as graphic card, motherboard type etc

3.2 The Proposed System

Autospec is a software that acts as a sticker for your PC. It presents a more wholistic environment in a window form where the system analyses and shows the statistics on every piece of hardware in your computer to the user; this includes CPU, Motherboard, RAM, Graphics card, Operating systems, Hard Disks, Optical Drives, Audio support. Additionally Autospec adds the performance of computer, so you can easily see if there is a problem and how you can resolve it and also an interface for checking for software updates.

The application is aimed at reducing or eliminating the shortcomings of the existing system. In this regard, **some of the benefits to be derived from the proposed system include:**

- ✓ Upgrade of the system memory, you can check how many memory slots your computer has and what memory's already installed.

- ✓ Use Autospec to quickly list the components, when buying/selling a PC. You can use Autospec to check that the computer has what the label says it has.
- ✓ Autospec has all the information on one easy-to-understand screen. There is no need to hunt around Windows
- ✓ Autospec gives you the option for checking for software update when connected to the internet.
- ✓ It also helps you to determine causes of overhead in your computer and possible solutions to improve the performance

3.3 Design Methodology

The methodology adopted in the analysis and design of this work is object-oriented paradigm and visual modeling throughout the development life cycles to foster better user communication and product quality.

3.3.1 Object-Oriented Analysis and Design

Object-oriented analysis

The purpose of any analysis activity in the software life-cycle is to create a model of the system's functional requirements that is independent of implementation constraints.

Common models used in OOA are use cases and object models. Use cases describe scenarios for standard domain functions that the system must accomplish.(Jacobsen et al, 1992)

3.4 Design Tool

The design tool used in this research is Unified Modelling Language (UML). The reason for adopting UML as a design tool in our software development is that it is graphical language for modelling software systems. UML consist of a number of diagrams that can be used to analyse, specify, construct, visualize, and document software designs. UML has diagrams that guides a developer at every stage of the application development process, from requirements gathering through design, and into coding, testing and deployment.

The Unified Modeling Language (UML) offers a way to visualize a system's architectural blueprints in a diagram including elements such as:

- Any activities (jobs)
- Individual components of the system
- And how they can interact with other software components.
- How the system will run
- How entities interact with others (components and interfaces)
- External user interface
- How the system is expected to be used.

Although originally intended solely for object-oriented design documentation, the Unified Modeling Language (UML) has been extended to cover a larger set of design documentation (as listed above),(Satish, 1997) and been found useful in many contexts.

3.4.1 Use case diagram

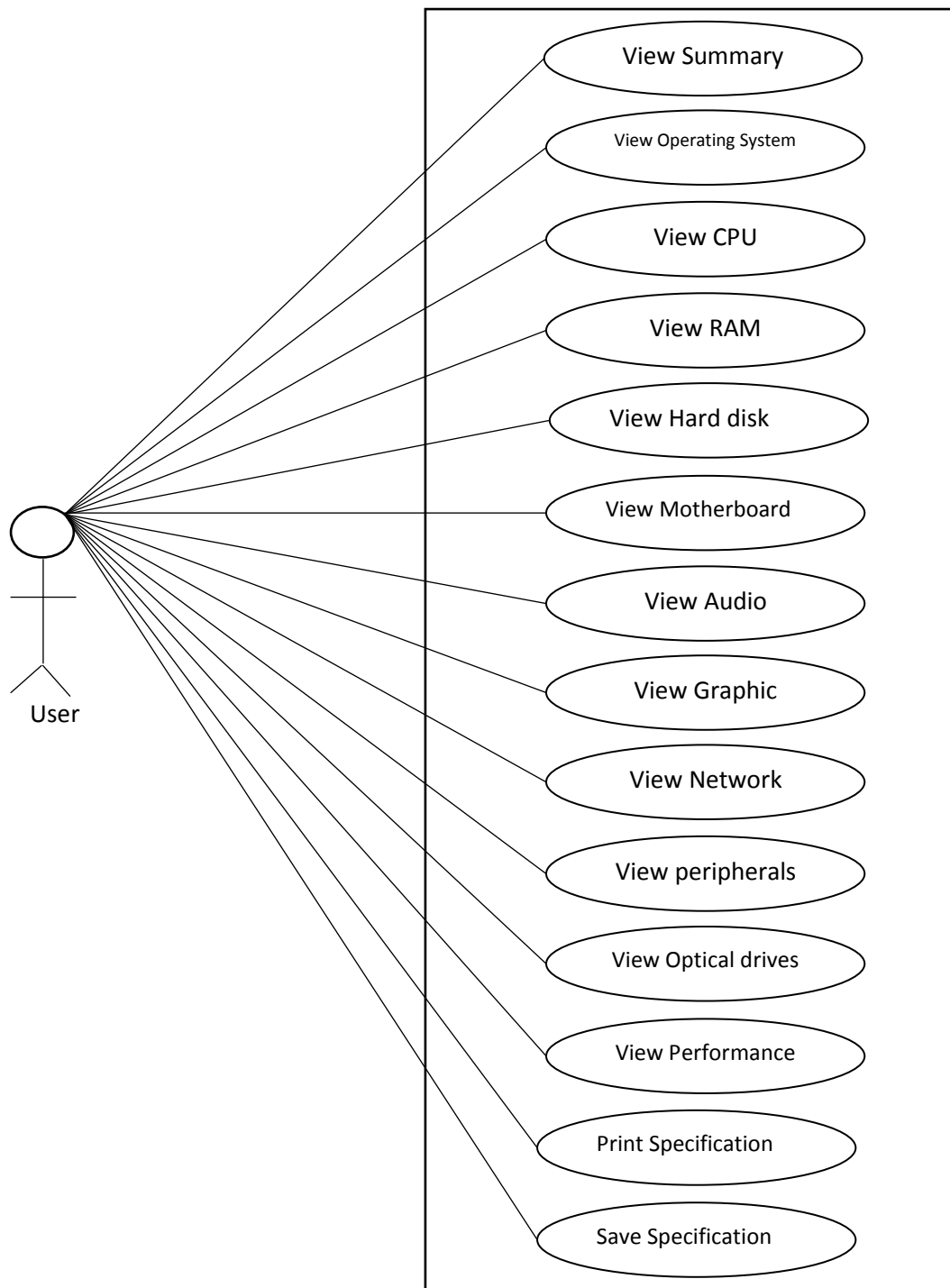


Fig. 3.1:AutoSpec Use Case Diagram

3.4.2 Class Diagram

Class diagram depicts the system's object structure. They show object classes that the system is composed of as well as the relationships between those object classes

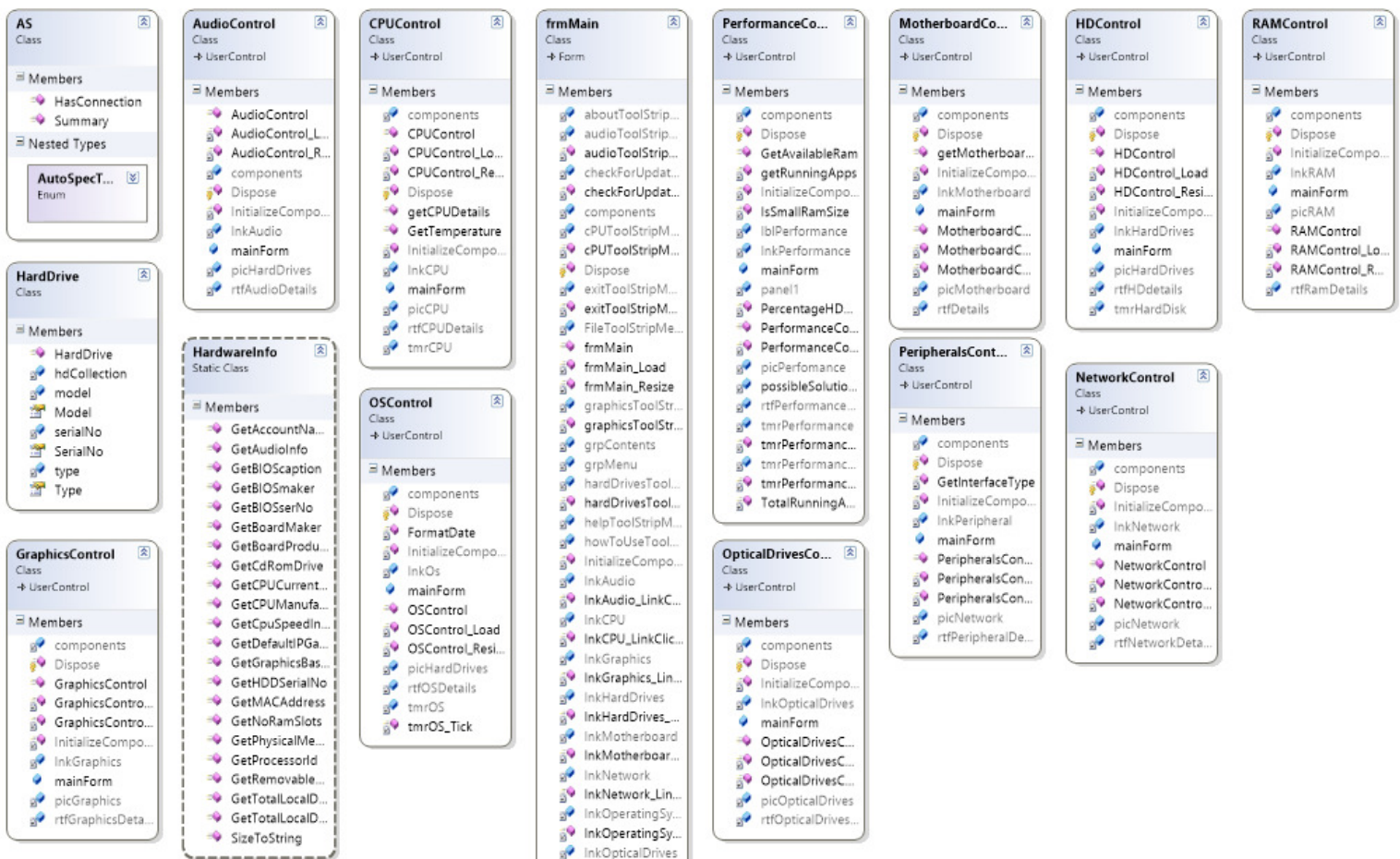


Fig. 3.2: Class diagram

3.5 Data Specification

The application has no database. It only makes use of file system to store data. The data is stored on user request. This can be done by clicking the File menu of the system and selecting **Save As Text File** option. The system specification is stored with the extension .txt which shows that it is an ordinary text file. The file name comes with the name of system specification type say OsControl.txt motherboardControl.txt etc. The file name follows the normal naming convention of MS file name.

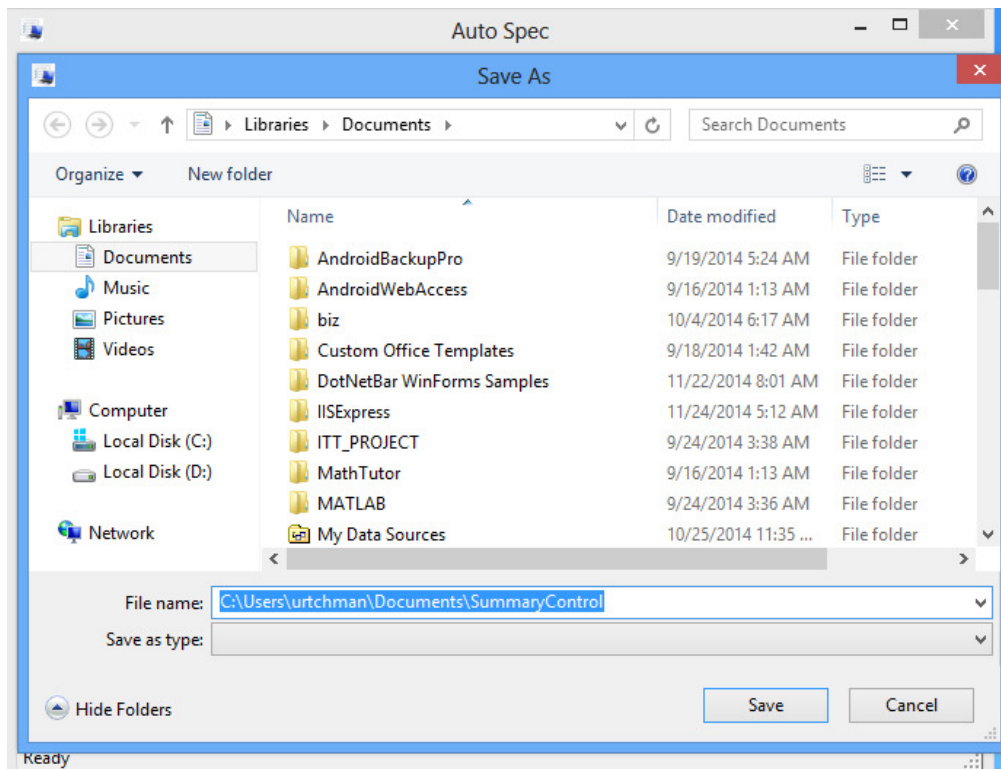


Fig. 3.3: Save Specification as text file

Chapter 4: **System Implementation**

4.0 Introduction

Implementation is the realization of an application, or execution of a plan, idea, model, design, specification, standard, algorithm, or policy. Systems implementation is concerned with the construction of the new system and the delivery of that system into production (that is, the day-to-day business or organization operation). In software development, system implementation covers choice of programming language used, the development environment, software testing and documentation.

The AutoSpec software was implemented with Visual Studio 2010 (the best Integrated Development - IDE). The IDE has so many things in it. The C# compiler is there, the MS SQL server and other tools.

4.1 Choice of Development Environment

4.1.1 Visual Studio .NET

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as web sites, web applications and web services. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight.

Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for source-control systems (like Subversion) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

Visual studio is user friendly and has many tools that a novice can use without been guided. Other IDEs like eclipse, NetBeans, dreamweaver etc. are imitating the structure of visual studio and this makes visual studio a pace therefore the best IDE in the world. This justifies why it was chosen for this project.

4.1.2 Choice of Programming Language

C# (pronounced as see sharp) is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure. C# is intended to be a simple, modern, general-purpose, object-oriented programming language. C# is designed for building robust and durable components to handle real- world situations.

C# and Java are world leading programming but C# is more flexible. It contains more functionalities, data types than Java. C# programs are developed with visual studio which happens to be the best IDE in the world. This means that it is to write program in C# than other none Microsoft based programming languages like Java, Python etc.

4.2 Implementation Architecture

The system was implemented based on the block diagram shown in fig. 4.1. AutoSpec has 12 system specification categories namely:

- (i) **Summary:** this contains quick summary of the system running the application.
- (ii) **Operating System:** this contains the details of the operating system in which the application is running. It gives every details of what an operating system has.
- (iii) **CPU:** this contains every information about the system's central processing unit.
- (iv) **RAM:** this contains information about the system physical memory.
- (v) **Motherboard:** the detailed information about the motherboard is contained in this section of the system specification.
- (vi) **Graphics:** the motherboard resolution and graphic characteristics are displayed in this section.

- (vii) **Hard Drives:** this contains information about the hard drives present in the system running this application.
- (viii) **Optical Drives:** this contains information about optical drives plugged into the system while this application is running. When this application is running, if any flash drive, modem or other drives are plugged, the software detects them and show their details.
- (ix) **Audio:** this contains the audio properties of the system running this application.
- (x) **Peripherals:** this gives details of all the peripherals of the system.
- (xi) **Network:** this contains the details of the system network and adapters.
- (xii) **Performance:** this contains the analysis of the system performance. It shows the reason why system is slow and suggests ways of optimizing the CPU usage.

All the system specification categories normally display in one central display unit. When a category is selected, the program removes what is on the system specification display unit and places the new category details on it.

Each of the specification category was implemented using userControl feature of visual studio. The flowchart of the entire system is as shown in fig. 4.2. The flowchart covers all the parts of the software application.

4.2.1 Main interface Implementation

The main interface was implemented using visual C# on .net framework of Microsoft visual studio 2010. The concept of pick and drop was used to arrange the controls used in the project. More graphical works were added to give the project shape. The main interface is actually a static one. It gives no room for user to input any kind of text as input. The only time the user is expected to input text is during saving of system specification. The application contains just one main form called frmMain. It is on top of this form that several user controls are called up and placed on the display unit of the main interface. The application also has two other small forms namely frmAbout and frmHelp. frmAbout contains information about the whole application while frmHelp contains instruction on how to use the application.

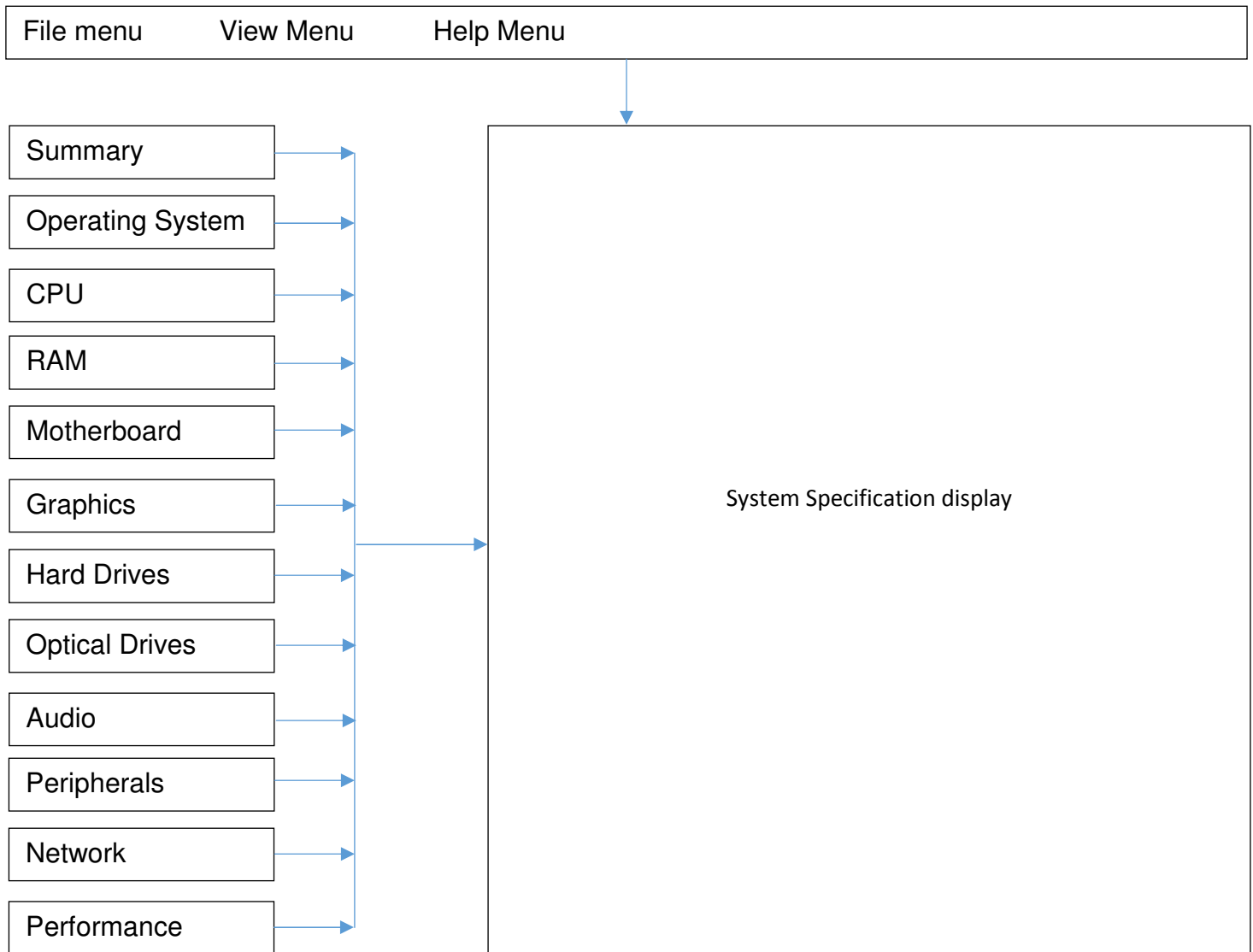


Fig. 4.0: System Block Diagram

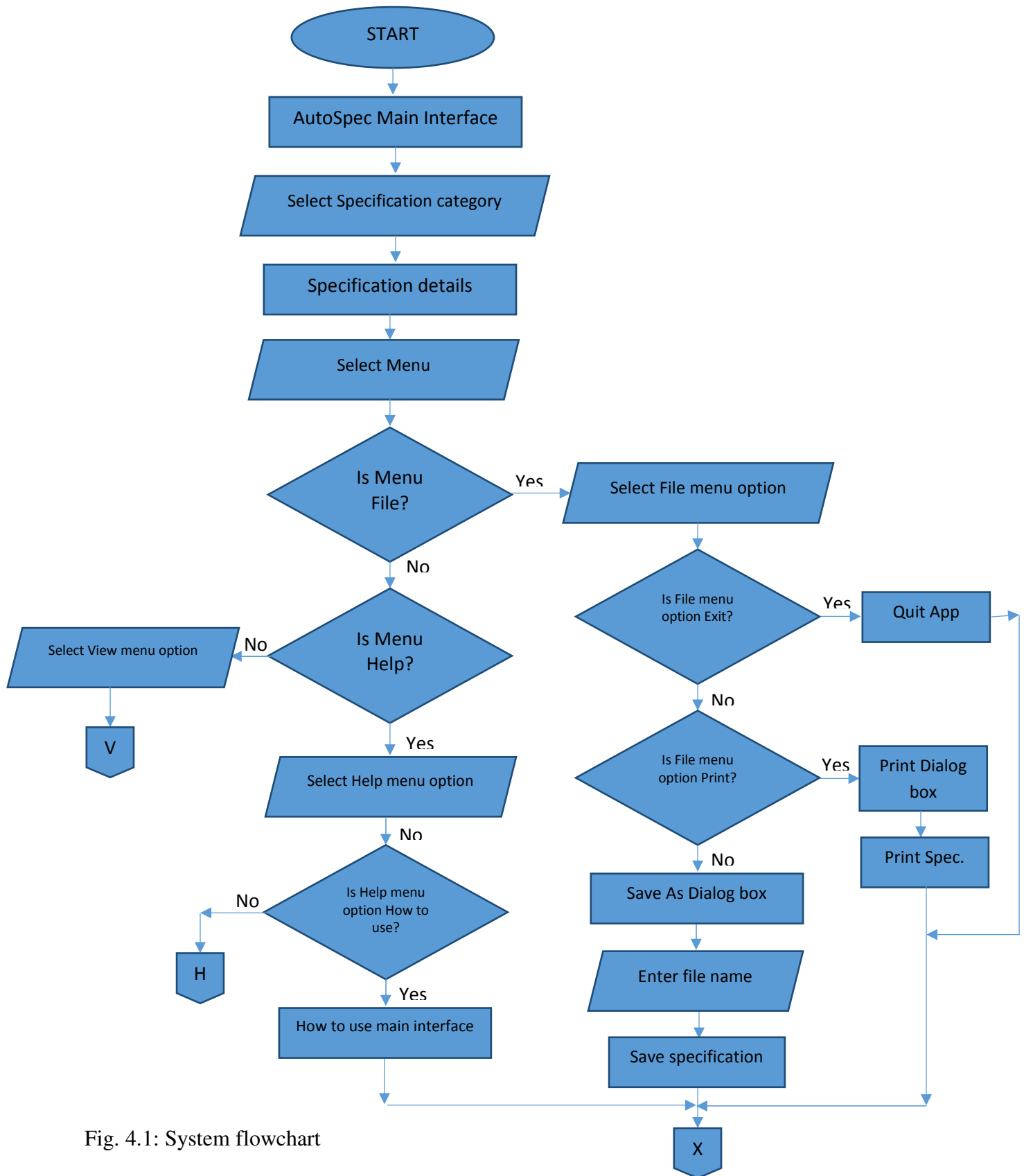


Fig. 4.1: System flowchart

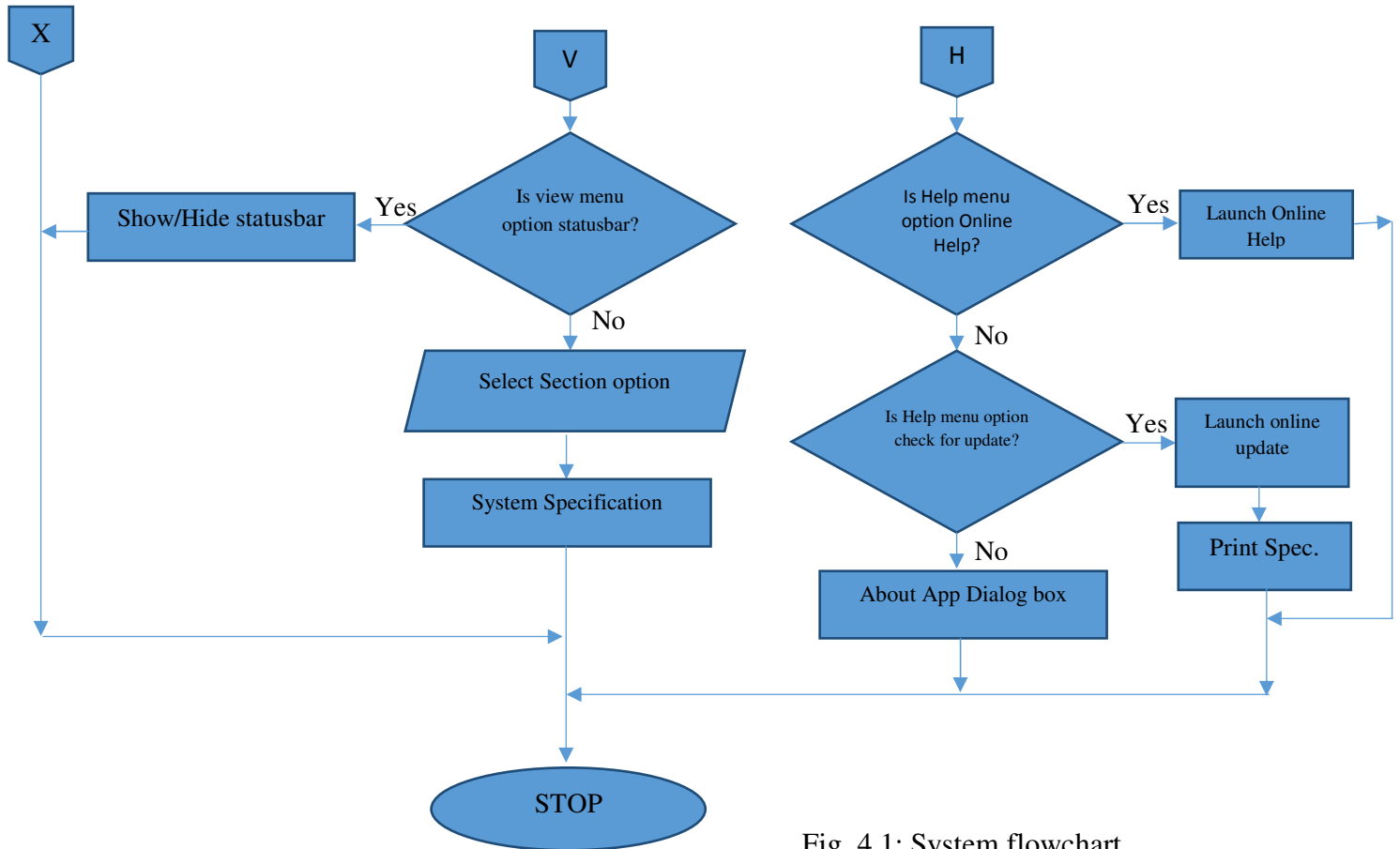


Fig. 4.1: System flowchart

4.2.1 Main interface Implementation

The main interface was implemented using visual C# on .net framework of Microsoft visual studio 2010. The concept of pick and drop was used to arrange the controls used in the project. More graphical works were added to give the project shape. The main interface is actually a static one. It gives no room for user to input any kind of text as input. The only time the user is expected to input text is during saving of system specification. The application contains just one main form called frmMain. It is on top of this form that several user controls are called up and placed on the display unit of the main interface. The application also has two other small forms namely frmAbout and frmHelp. frmAbout contains information about the whole application while frmHelp contains instruction on how to use the application.

4.2.2 Input Implementation

It is worthy to note that this system has no input fields at all. This means that for a user to see an output, no input is needed. The only thing that looks like input is the specification category. For the user to see the output of the system, one of the categories must be clicked. So the input to the system is only made by clicking not through the keyboard as obtained in other software application. As mentioned earlier, the only time the user enters input from the keyboard is when the user is trying to save output of system specification. This cannot be said to be main input because it only happens during saving of system specification. Moreover it is not required to view the output of the system so it is kind of tertiary input.

4.2.3 Output Implementation

The output of the system was implemented using a special control called **User Control**. User Control allows the programmer to customize already existing controls to match his own task. Each of the 12 categories of the system specification is a user control. They all stand alone and are called up each time they are needed especially when the link is clicked. The main screen contains a panel where each of the 12 controls are placed. The summary of the 12 controls are summarized in Table 4.1.

Table 4.1: List of output screen controls

U s e r C o n t r o l	F u n c t i o n
A u d i o C o n t r o l	Displays audio details of the system
C P U C o n t r o l	Displays CPU details
G r a p h i c s C o n t r o l	Displays Graphics details
H D C o n t r o l	Displays Hard Drives details
M o t h e r b o a r d C o n t r o l	Displays motherboard details
N e t w o r k C o n t r o l	Displays network details
O p t i c a l D r i v e s C o n t r o l	Displays optical drives details
O S C o n t r o l	Displays operating system details
P e r f o r m a n c e C o n t r o l	Displays the system performance
P e r i p h e r a l s C o n t r o l	Displays peripherals details
R A M C o n t r o l	Displays RAM details
S u m m a r y C o n t r o l	Displays the system specification quick summary

4.3 Software Testing

System testing is the process of executing program with the primary aim of viewing the output and checking for errors. Autospec was meticulously tested in order to determine the stability of the software. During the first phase of the test, the software was run on another system (computer) apart from the one it was coded with. This worked fine but a lot of bugs were discovered as a result. Through the use of some tools in Visual C Sharp such as the watch and immediate window, we were able to identify and fix those faulty parts of the program (software). such as network error handler front size and wrong date format.

Further phase of the testing focused on user friendliness. We optimized the user interface to make it more users friendly and easy to use

4.4 Screen Shots of Demos

When the application is launched, the very first page displays the application summary page. It shows a quick summary of the system specifications.

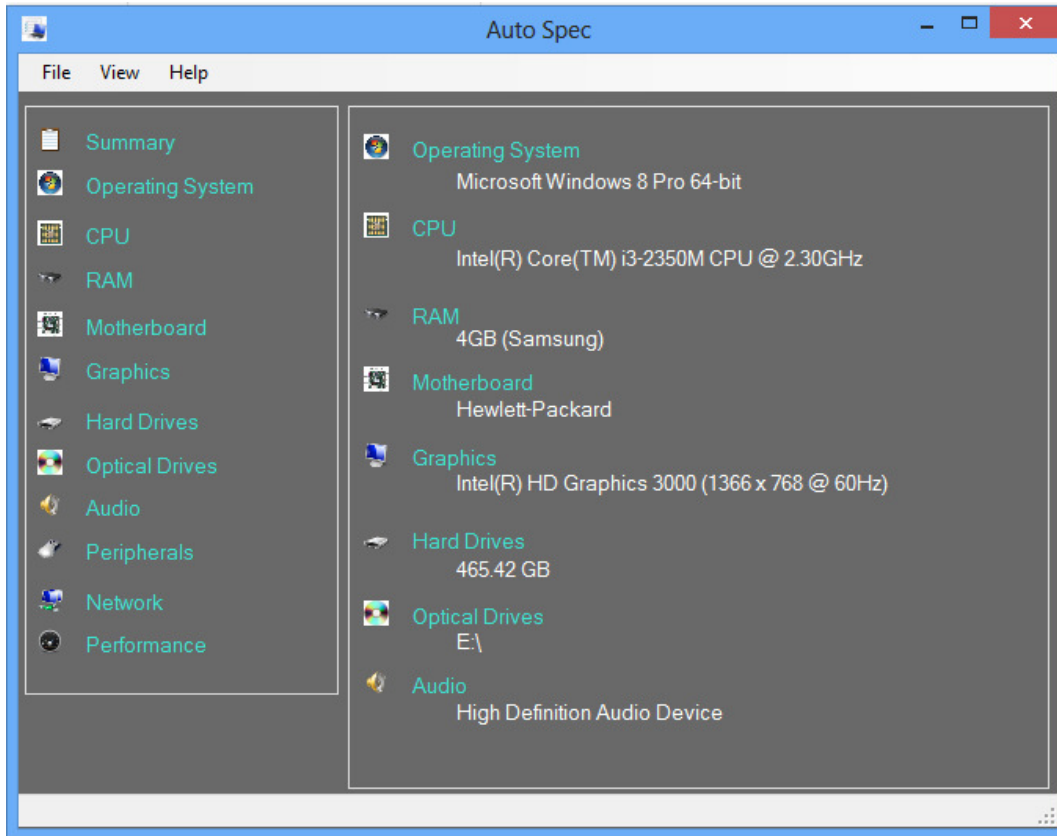


Fig. 4.3: Auto Spec summary screenshot

The left hand of the application is a list of different system specifications like operating system, CPU, RAM, motherboard, Graphics, Hard drives, optical drives audio, peripherals, network and performance. Any of the specification that is clicked brings out the details. The screenshot in fig. 4.3 shows the operating system details.

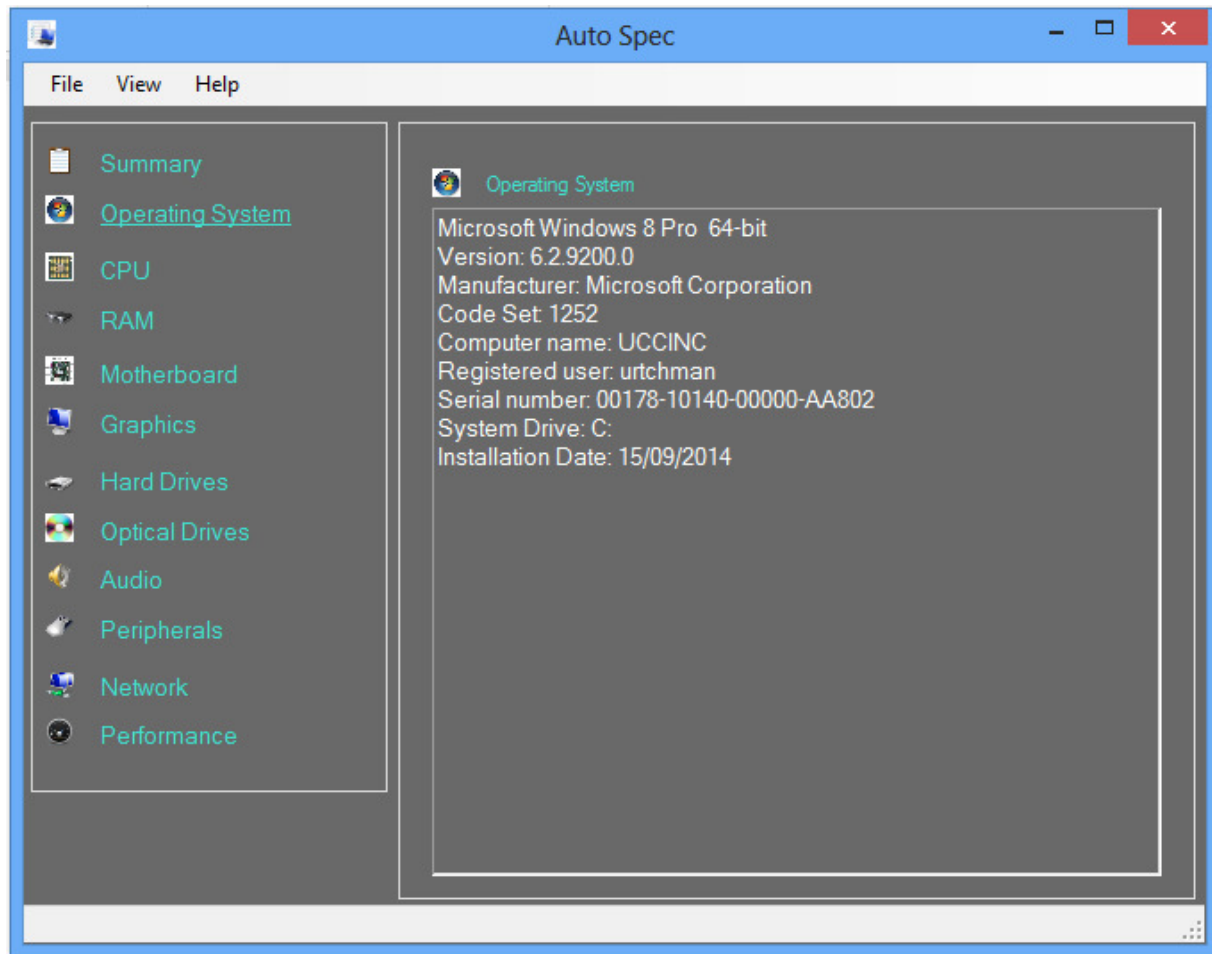


Fig. 4.4: Operating System details screenshot:

when the user selects operating system, it displays all the information about the operating system such as Bit rating, OS type and installation date e. t. c.

When the CPU is selected, the details of the CPU displays as shown in fig. 4.5. The details displayed is system dependent. These

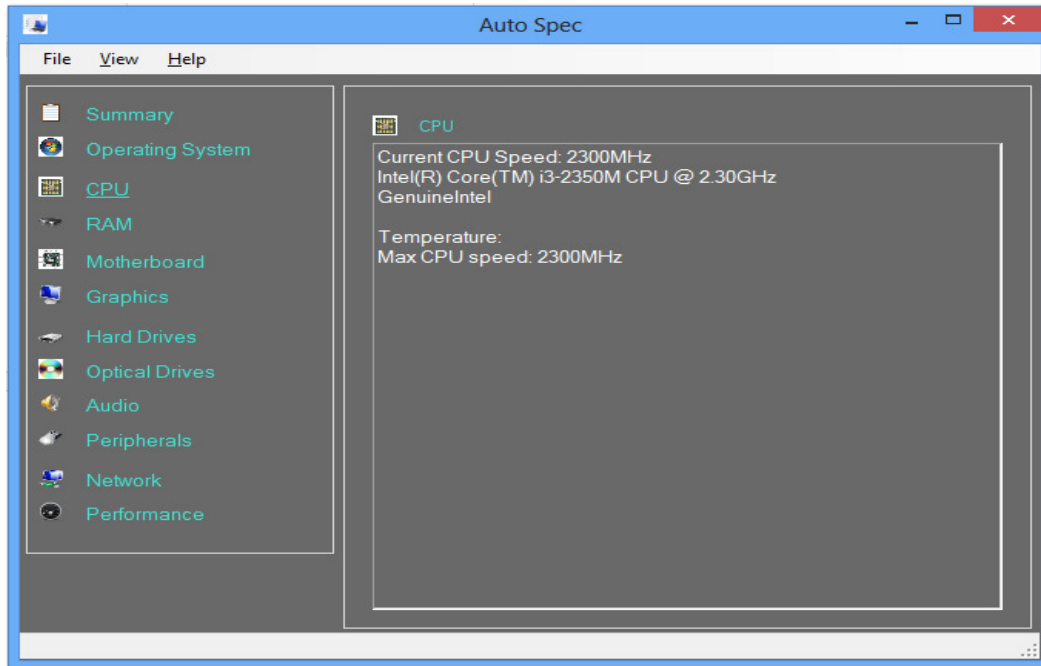


Fig. 4.5: CPU details screenshot

In the same vein when the RAM is selected, the details of the system random access memory is displayed as shown in fig. 4.6

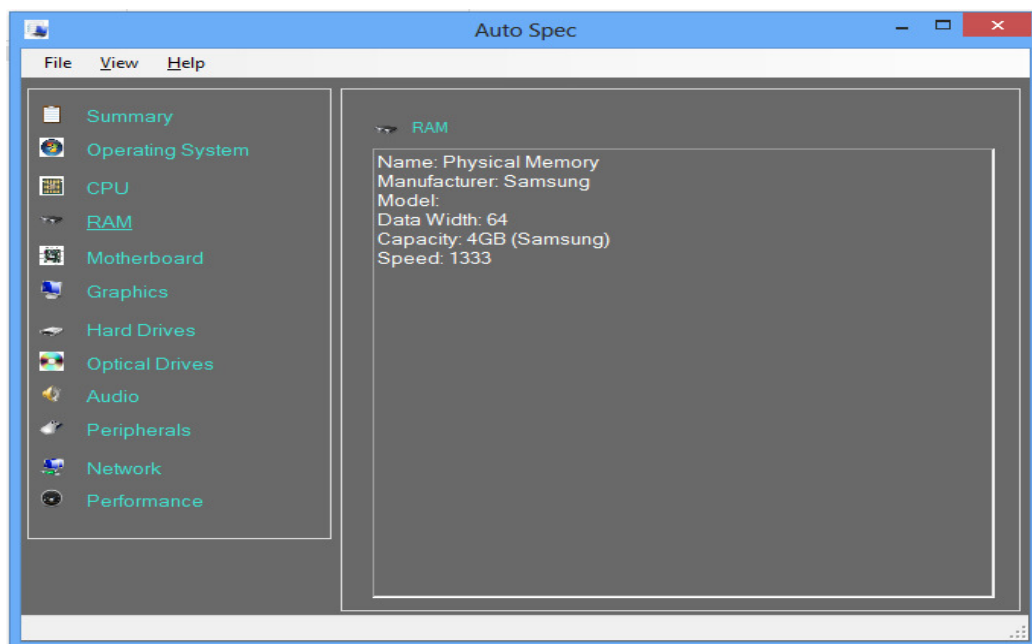


Fig. 4.6: RAM details screenshot

When the Motherboard is selected, the comprehensive details of the system motherboard is displayed as shown in fig. 4.6.

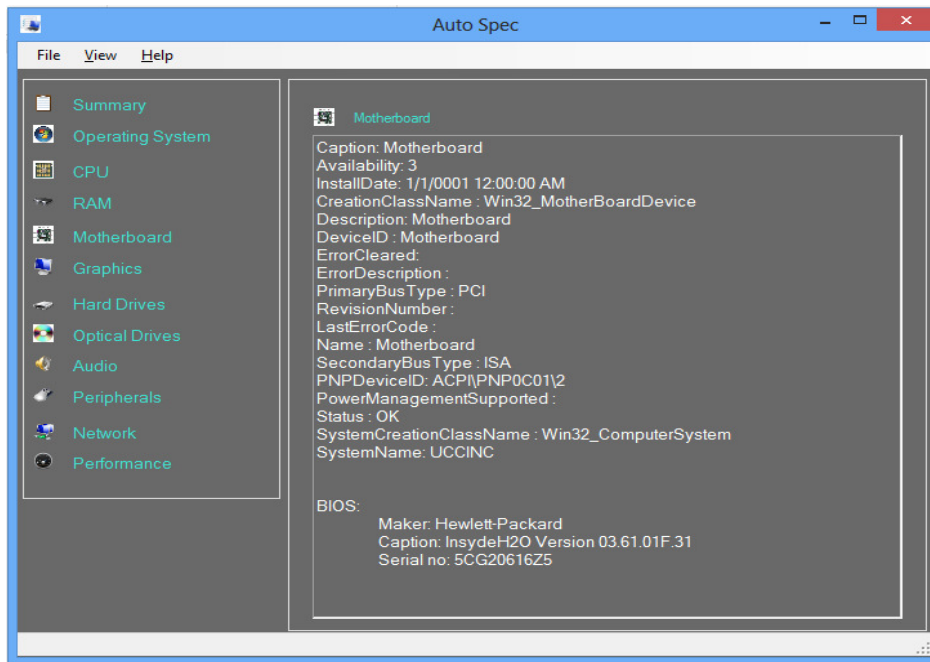


Fig. 4.7: Motherboard details screenshot.

When the graphicsdisplay is selected, it shows the monitor specifications such as the monitor width and height etc.

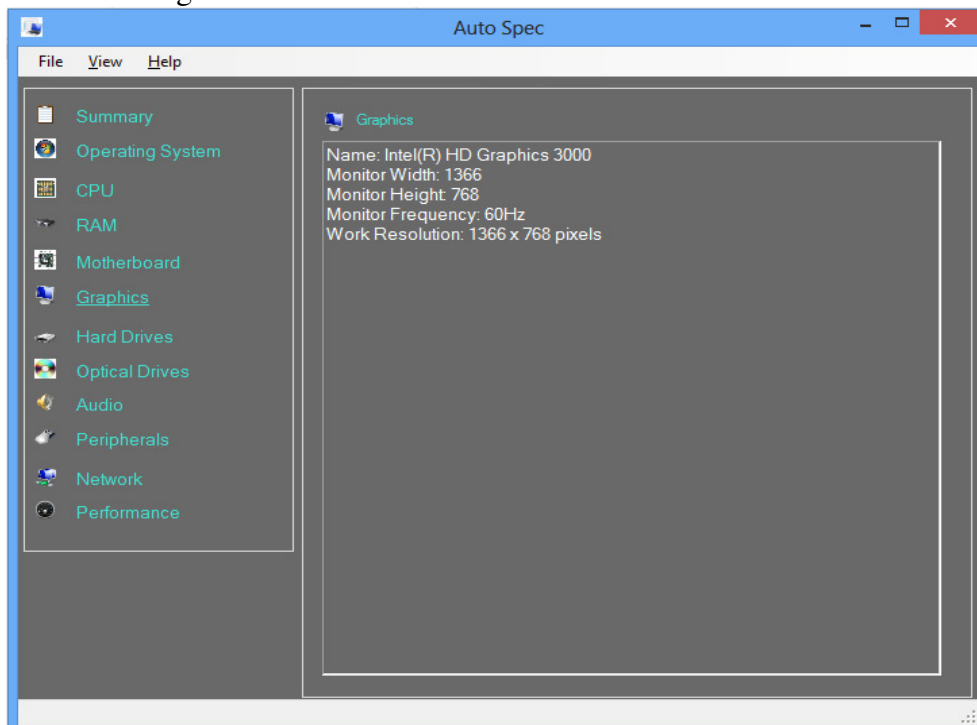


Fig. 4.8: Graphics display details screen.

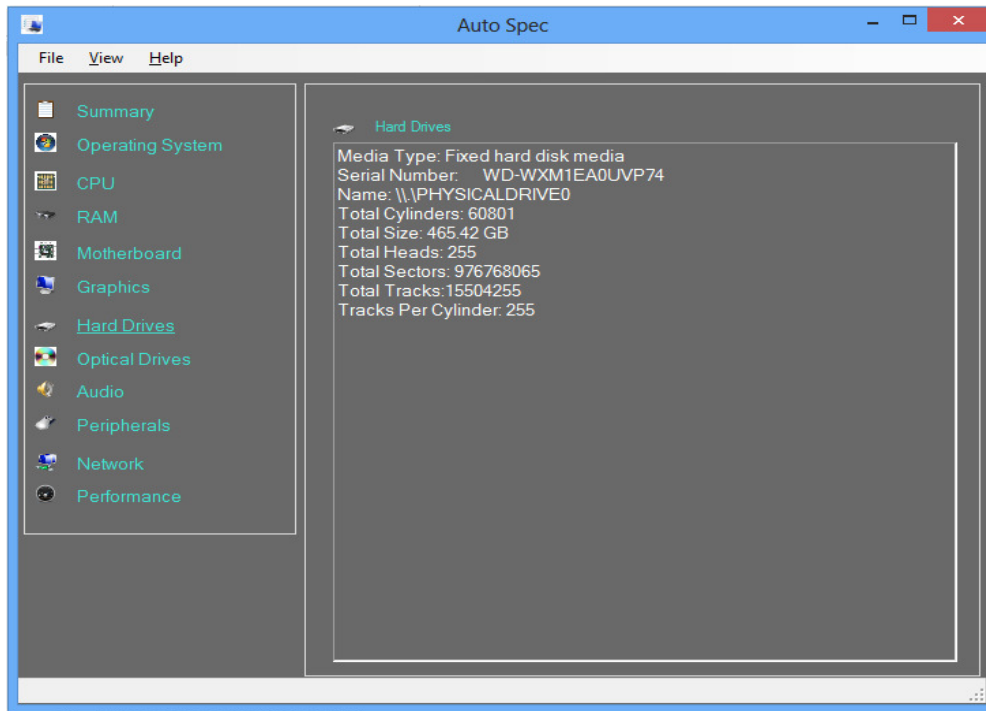


Fig. 4.9: Hard Drives details screenshot

when the Hard Drives is selected, it displays all the information about the disk such the total size, total heads and sectors etc.

When the optical drive is selected, it displays all the drives connected to the system. As at the time when the system was tested, one optical drive was plugged into the system and the system was able to detect and display the details as shown in fig. 4.9.

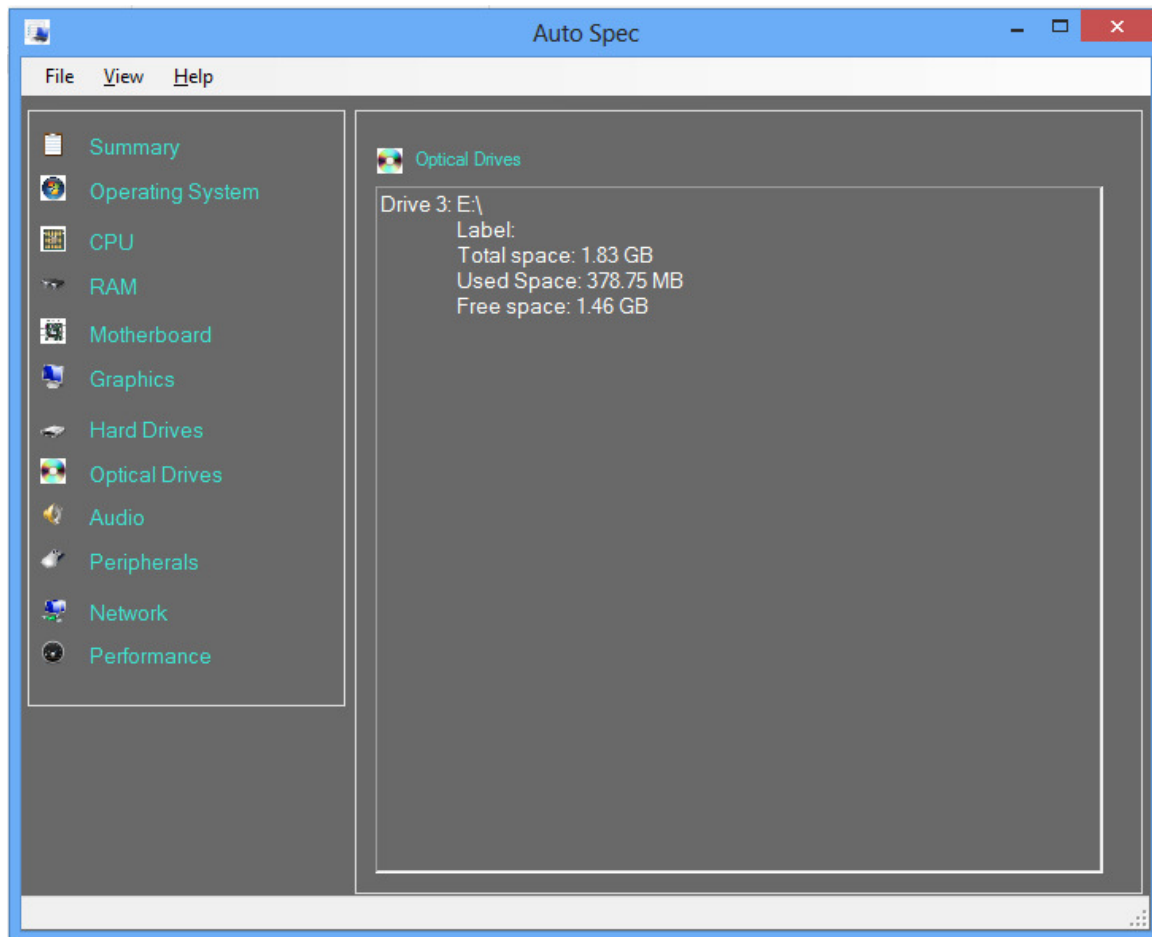


Fig. 4.10: Optical Drives details screenshot

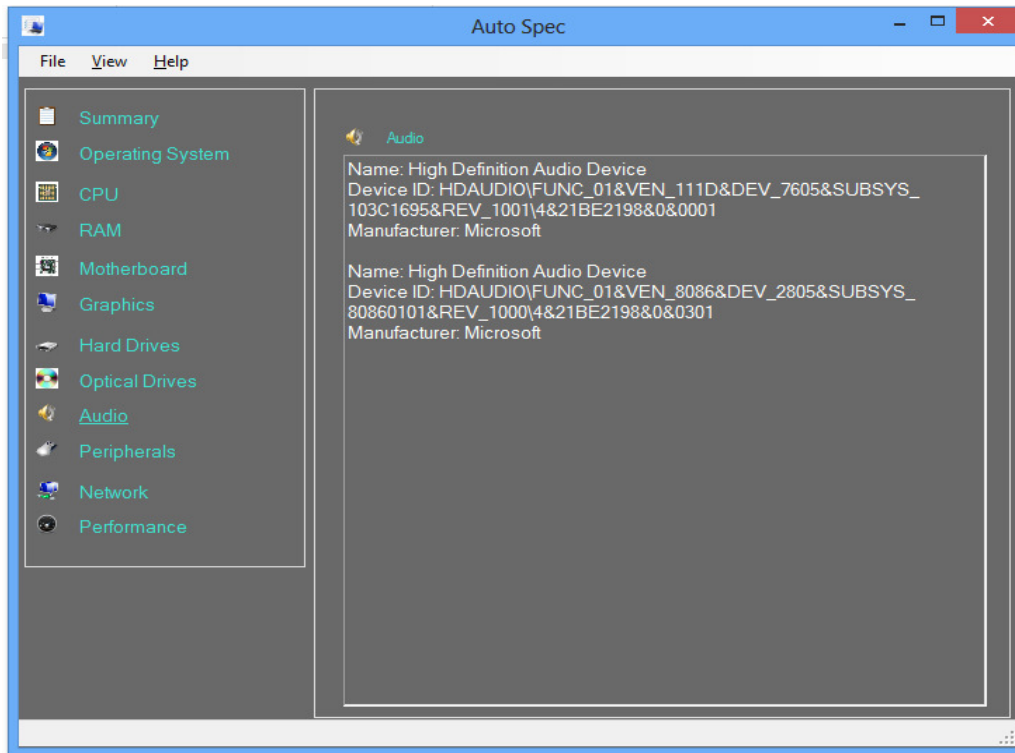


Fig. 4.11: Audio details screenshot:

when audio is selected, it displays information related to the system audio such name, device ID e. t. c.

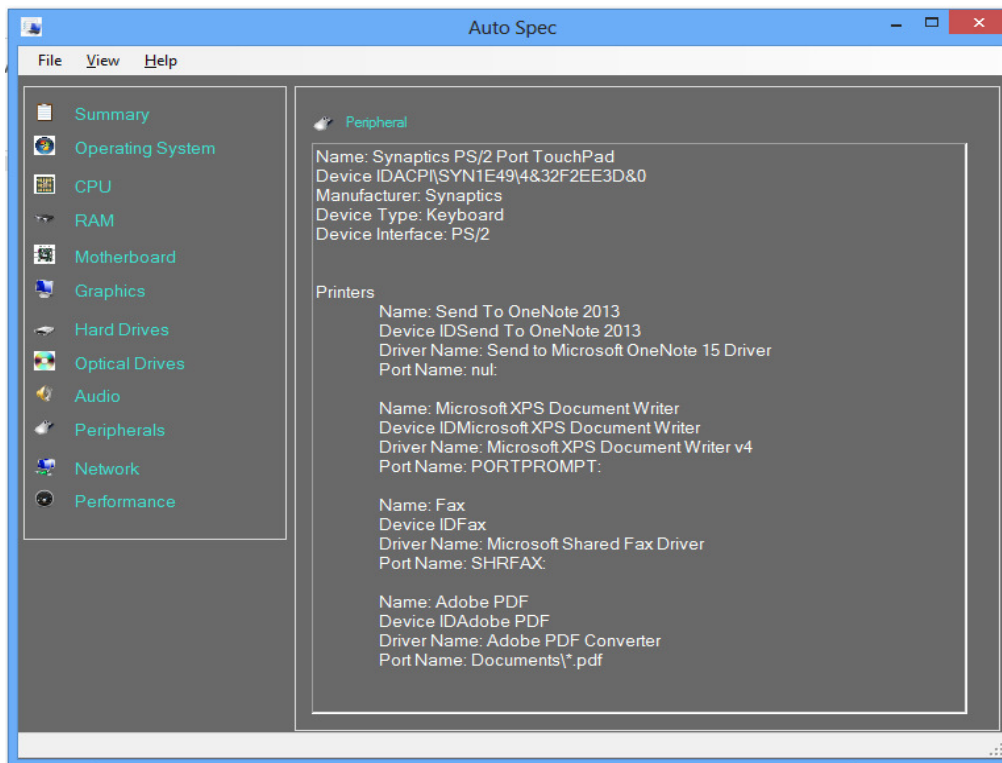


Fig. 4.12: Peripheral details screenshot:

When perisherial is selected, it displays all the information related to peripheral devices such as printer, keyboard e. t. c.

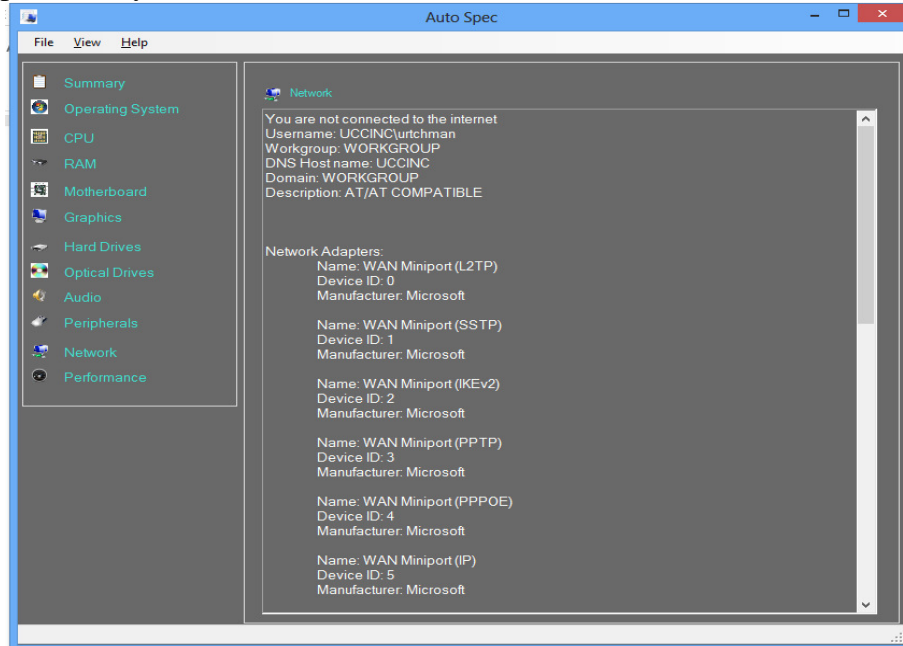


Fig. 4.13: Network details screenshot:

When Network is selected, it displays all the information related to the Networks such as connection status, Workgroup and DNS Host name etc.

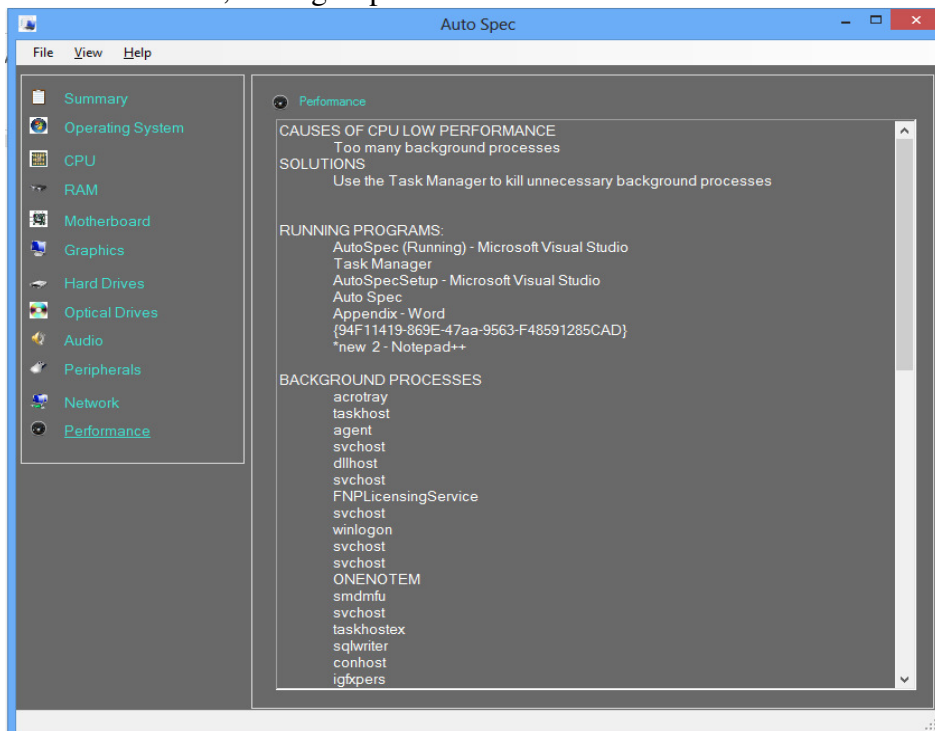


Fig. 4.14: Performance details screenshot:

when performance is selected, it show all the running process, background process, causes of overhead and possible solutions

4.5 Documentation

4.5.1 Hardware Requirement

Below are the minimum hardware requirements to run the application

- RAM: 1GB
- Processor speed of 1.4GHz
- Hard disk: 10GB
- Resolution: 800 by 600 pixels

4.5.2 Software Requirement

The software is .NET framework based which means that it can only run on Microsoft operating system. Below are the minimum software requirements to run this application:

- Operating system: Windows XP service pack 3, Windows vista, Windows 7, Windows8.
- .NET framework of version 4.0

4.5.3 User Manual

The application usage is a very simple and straight forward one. To use the application, a simple installation is required. Simply open the AutoSpec CD and double click the setup.exe file and follow the installation procedure to install the software. After installation, launch the application from all programs. The specification categories are arranged at the left hand side of the application, click any of them to view its details.

To save any of the specification details, click **File** menu then click **Save As Text File** and enter the name of the file or simply press **Enter** key because the application always come with a file name by default.

The specification details can also be printed out. To do this, simply click **File** menu and click **Print** option. The AutoSpec software is also published online for updating. If you want to update your version, simply click **Help** menu and select **Check for update** option. A web browser will

be launched for you automatically. You will see a link on the browser to either update or message telling you that there is no update available.

4.5.4 Source code listing

The system is made up of several controls with each control presented as a separate class. There are also other classes such as AS, frmMain and HardwareInfo. for source code listing of this See appendix (A to O).

Chapter 5: **Summary and Conclusion**

5.0 Summary

AutoSpec is an application that helps both experts and novice to retrieve system specification without having to move from application to another. AutoSpec has the whole system properties in one place and therefore helps the analysts to have full view of what the system is made up of.

One of the greatest achievements of this application is that a novice can see the stuff a PC is made of without being taught how to do so. In traditional system specifications, one has to move from one part of the system, from one application of the system to another just to get different information about the system. This application has made it possible to get the whole information with just a click of the mouse.

Another achievement of the application is its ability to analyse system performance, detect why the system is slow, and suggest some important things to be done in order to fix them. The application equally shows all the running programs and background processes in the system and how they affect the performance of such system.

The application is a light weight one so it matches any system running it.

5.1 Conclusion

System specification is one thing that a user of a PC finds difficult in viewing and analysing. In fact majority of computer users don't even know where to view the system specification. This application has brought together every basic information a user needs to know about a personal computer. With the achievements of this project work, it can be concluded that this software application is what computer users need to analyse their systems and get full information on the system specifications and performance without having to bring in computer experts.

5.3 Recommendations

Research work is a continuous process and many things can be added to an existing system and the addition becomes a research on its own. Due to the numerous benefits in this software application and its openness to improvement, we hereby recommend that:

1. Computer users should be advised to use the application.
2. Students should be taught system analysis and computer maintenance so as to be able to improve this work as technologies change/improve.

REFERENCES

- Arnold O. Allen (1994): “Computer Performance Analysis with Mathematics”. Academic Press, Roseville California,. pp 1. ISBN 978-0-12-051070-2.
- Agarwal, A.; Levy, M.,(2007): ‘The KILL Rule for Multicore’, Design Automation Conference, DAC '07. 44th ACM/IEEE, vol., no.4, pp.750-753, 4-8 June 2007
- Chapman, G. Justand R. Van de Pass,(2008): ‘using OpenMP, Portable Shared Memory Parallel Programming’, Massachusetts Institute of Technology, pp.4, 16-35
- Conallen, Jim (2000).“Building Web Applications with UML”. Addison Wesley. p.147. ISBN 0201615770.
- Carpenter R. E.,(2007): “Comparing Multi-Core Processors for Server Virtualization”, Intel Corporation.http://www.multicoreinfo.com/research/papers/whitepapers/multicore_virtualization.pdf
- Domeika M., (2009): “Evaluating the Performance of Multi-Core Processors”,<http://www.embedded.com/design/embedded/4008794/Evaluating-the-performance-of-multi-core-processors-Part--1>
- Faxen K.,Bengtsson C.,Bronsson M., Grahn H.,Hagersten E., Jonsson B., Kessler C.,Lisper B., Stenstrom P., Svensson B.,(2008)"Multicore Computing-The State of Art", p.3. URL: <http://eprints.sics.se/3546/>
- Jacobsen, Ivar; Magnus Christerson, Patrik Jonsson and Gunnar Overgaard (1992). “Object Oriented Software Engineering”. Addison-Wesley ACM Press. pp. 77–79. ISBN 0-201-54435-0.
- Jain R,(1991): “The Art of Computer System Performance Analysis” , Multicore Processor Performance Analysis - A Survey. Wiley- Interscience, New York, NY, April 1991, ISBN: 0471503361. Page 7 of 9

- Kayi A., Yao W., EL-Ghazawi T. and Newby G.(2007): ‘‘Experimental Evaluation of Emerging Multi-core Architectures’’, Parallel and Distributed Processing Symposium, IEEE International.
<http://cecs.uci.edu/~papers/ipdps07/pdfs/PMEO-PDS-21-paper-1.pdf>
- Mario Adolfo Zavala (2009): ‘‘High Speed Computing, Performance Analysis’’ sun blue prints, Arizona US, pp. 9-10.
- Monchiero M., Canal R. and Gonzalez A., (2006): ‘‘Design Space Exploration for Multicore Architectures:A Power/Performance/Thermal View’’, Proceedings of the 20th annual international conference on Supercomputers, ACM New York, NY, ISBN:1-59593-282-8, Pages 177 &186.
- Pase D. M. and Eckl M. A.,(2005) :‘A Comparison of Single-Core and Dual-Core Opteron Processor Performance for HPC’ , Technical report, IBM Developer Works, IBM Corporation.
ftp://ftp.support.lotus.com/eserver/benchmarks/wp_Dual_Core_072505.pdf
- Prinslow G.,(2011): ‘‘Overview of Performance Measurement and Analytical Modeling Techniques for Multi-core Processors’’,
<http://www.cse.wustl.edu/~jain/cse567-11/ftp/multcore/index.html>
- Ruud van der Pas,(2002): ‘‘High Performance Computing, Memory Hierarchy in cache-based systems’’, Sun blue prints,Arizona, pp. 2-3.November 2002.
 URL: <http://www.sun.com/blueprints/1102/817-0742.pdf>
- Sharma A., Kumble M., Moktali P. R.and Siri H. (2009): ‘‘Performance analysis of Multicore Systems’’, Intel, March <http://software.intel.com/en-us/articles/performanceanalysis-of-multicore-systems-4>
- West P.,(2008): Core Monitors: Monitoring Performance in Multicore Processors’, Master Thesis, The Florida State University- Computer Science Department.
- Wikipedia: "Moore's Law", http://en.wikipedia.org/wiki/Moore's_law
 Accessed on 12th may 2014.

Appendix A

AS class source code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Management;

namespace AutoSpec
{
    class AS
    {
        public static string Summary(AutoSpecType ast)
        {
            return "";
        }

        public enum AutoSpecType
        {
            Ram,
            Cpu,
            Motherboard,
            Graphics,
            Audio,
            Network,
            HardDrives,
            OpticalDrives,
            Peripherals,
            OS,
            Performance
        }

        public static bool HasConnection()
        {
            try
            {
                System.Net.IPHostEntry i = System.Net.Dns.GetHostEntry("www.google.com");
                return true;
            }
            catch
            {
                return false;
            }
        }
    }
}
```

Appendix B

AudioControl source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;

namespace AutoSpec
{
    public partial class AudioControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public AudioControl()
        {
            InitializeComponent();

            private void AudioControl_Load(object sender, EventArgs e)
            {
                ManagementObjectSearcher mo = new ManagementObjectSearcher("select * from Win32_SoundDevice");

                foreach (ManagementObject soundDevice in mo.Get())
                {
                    String deviceId = soundDevice.GetPropertyValue("DeviceID").ToString();
                    String name = soundDevice.GetPropertyValue("Name").ToString();

                    rtfAudioDetails.Text += "Name: " + name + "\nDevice ID: " + deviceId +
                    "\nManufacturer: " +
                    soundDevice.GetPropertyValue("Manufacturer") + "\n\n";
                }
                mainForm.textToSave = "AUDIO\n" + rtfAudioDetails.Text;
            }

            private void AudioControl_Resize(object sender, EventArgs e)
            {
                rtfAudioDetails.Width = this.Width - 24;
                rtfAudioDetails.Height = this.Height - 40;
            }
        }
    }
}
```


Appendix C

CPU control source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;

namespace AutoSpec
{
    public partial class CPUControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public CPUControl()
        {
            InitializeComponent();

            public string getCPUDetails()
            {
                string cpu = "";
                using (ManagementObjectSearcher win32Proc = new ManagementObjectSearcher("select * from Win32_Processor"),
                    win32CompSys = new ManagementObjectSearcher("select * from Win32_ComputerSystem"),
                    win32Memory = new ManagementObjectSearcher("select * from Win32_PhysicalMemory"))
                {
                    foreach (ManagementObject obj in win32Proc.Get())
                    {
                        string clockSpeed = obj["CurrentClockSpeed"].ToString(); //retrieves the current speed of the cpu
                        string procName = obj["Name"].ToString(); //retrieves the name of the cpu
                        string manufacturer = obj["Manufacturer"].ToString(); //retrieves the manufacturer of the cpu
                        string version = obj["Version"].ToString(); //retrieves the version of the cpu
                        cpu += "Current CPU Speed: " + clockSpeed + "MHz\n" + procName + "\n" + manufacturer + "\n" + version + "\nTemperature: " + GetTemperature() + "\nMax CPU speed: " + obj["MaxClockSpeed"].ToString() + "MHz";
                    }
                }
                return cpu;
            }

            private void CPUControl_Load(object sender, EventArgs e)
            {
                rtfCPUDetails.Text = getCPUDetails(); //displays the cpu details
                mainForm.textToSave = "CPU\n" + rtfCPUDetails.Text;
            }
        }
    }
}
```

```

//This method retrieves the cpu temperature
publicstring GetTemperature()
{
string strTemp = "";
ManagementObjectSearcher searcher = newManagementObjectSearcher(@"root\WMI", "SELECT *
FROM MSAcpi_ThermalZoneTemperature");
/*foreach (ManagementObject obj in searcher.Get())
{
Double temp = Convert.ToDouble(obj["CurrentTemperature"].ToString());
temp = (temp - 2732) / 10.0;
strTemp += temp;
}*/
return strTemp;
}

privatevoid CPUControl_Resize(object sender, EventArgs e)
{
//Makes the cpu details control to resize its size as the main control resizes
rtfCPUDetails.Width = this.Width - 24;
rtfCPUDetails.Height = this.Height - 40;
}
}

```

Appendix D

Main form source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace AutoSpec
{
    public partial class frmMain : Form
    {
        public string textToSave;
        public string summary;
        public frmMain()
        {
            InitializeComponent();
        }

        private void summaryToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LoadSummary();
        }

        private void operatingSystemToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LoadOS();
        }

        private void cPUToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LoadCPU();
        }

        private void rAMToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LoadRAM();
        }

        private void motherboardToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LoadMotherboard();
        }

        private void graphicsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LoadGraphics();
        }

        private void hardDrivesToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LoadHD();
        }
    }
}
```

```

privatevoid opticalDrivesToolStripMenuItem_Click(object sender, EventArgs e)
{
    LoadOpticalDrives();
}

privatevoid audioToolStripMenuItem_Click(object sender, EventArgs e)
{
    LoadAudio();
}

privatevoid peripheralsToolStripMenuItem_Click(object sender, EventArgs e)
{
    LoadPeripherals();
}

privatevoid networkToolStripMenuItem_Click(object sender, EventArgs e)
{
    LoadNetwork();
}

privatevoid lnkSummary_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    summaryToolStripMenuItem_Click(sender, e);
}

privatevoid lnkOperatingSystem_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    operatingSystemToolStripMenuItem_Click(sender, e);
}

privatevoid lnkCPU_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    cPUToolStripMenuItem_Click(sender, e);
}

privatevoid lnkRAM_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    rAMToolStripMenuItem_Click(sender, e);
}

privatevoid lnkMotherboard_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    motherboardToolStripMenuItem_Click(sender, e);
}

privatevoid lnkGraphics_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    graphicsToolStripMenuItem_Click(sender, e);
}

privatevoid lnkHardDrives_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    hardDrivesToolStripMenuItem_Click(sender, e);
}

privatevoid lnkOpticalDrives_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{

```

```

        opticalDrivesToolStripMenuItem_Click(sender, e);
    }

privatevoid lnkAudio_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    audioToolStripMenuItem_Click(sender, e);
}

privatevoid lnkPeripherals_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    peripheralsToolStripMenuItem_Click(sender, e);
}

privatevoid lnkNetwork_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    networkToolStripMenuItem_Click(sender, e);
}

privatevoid picSummary_Click(object sender, EventArgs e)
{
    summaryToolStripMenuItem_Click(sender, e);
}

privatevoid picOperatingSystem_Click(object sender, EventArgs e)
{
    operatingSystemToolStripMenuItem_Click(sender, e);
}

privatevoid picCPU_Click(object sender, EventArgs e)
{
    cPUToolStripMenuItem_Click(sender, e);
}

privatevoid picRAM_Click(object sender, EventArgs e)
{
    rAMToolStripMenuItem_Click(sender, e);
}

privatevoid picMotherboard_Click(object sender, EventArgs e)
{
    motherboardToolStripMenuItem_Click(sender, e);
}

privatevoid picGraphics_Click(object sender, EventArgs e)
{
    graphicsToolStripMenuItem_Click(sender, e);
}

privatevoid picHardDrives_Click(object sender, EventArgs e)
{
    hardDrivesToolStripMenuItem_Click(sender, e);
}

privatevoid picOpticalDrives_Click(object sender, EventArgs e)
{
    opticalDrivesToolStripMenuItem_Click(sender, e);
}

```

```

privatevoid picAudio_Click(object sender, EventArgs e)
{
    audioToolStripMenuItem_Click(sender, e);
}

privatevoid picPeripherals_Click(object sender, EventArgs e)
{
    peripheralsToolStripMenuItem_Click(sender, e);
}

privatevoid picNetwork_Click(object sender, EventArgs e)
{
    networkToolStripMenuItem_Click(sender, e);
}

privatevoid frmMain_Resize(object sender, EventArgs e)
{
    grpContents.Width = this.ClientSize.Width - 215;
    grpContents.Height = this.ClientSize.Height - 50;
}

privatevoid picSummary_MouseHover(object sender, EventArgs e)
{
    Cursor = Cursors.Hand;
}

privatevoid picSummary_MouseLeave(object sender, EventArgs e)
{
    Cursor = Cursors.Default;
}

privatevoid frmMain_Load(object sender, EventArgs e)
{
    LoadSummary();
}

publicvoid LoadSummary()
{
    SummaryControl summary = newSummaryControl();
    summary.mainForm = this;
    pnlContents.Controls.Clear();
    summary.Dock = DockStyle.Fill;
    pnlContents.Controls.Add(summary);
}

publicvoid LoadOS()
{
    OSControl osCtrl = newOSControl();
    osCtrl.mainForm = this;
    pnlContents.Controls.Clear();
    osCtrl.Dock = DockStyle.Fill;
    pnlContents.Controls.Add(osCtrl);
}

publicvoid LoadCPU()
{
    CPUControl cpuCtrl = newCPUControl();
    cpuCtrl.mainForm = this;
}

```

```

pnlContents.Controls.Clear();
    cpuCtrl.Dock = DockStyle.Fill;
pnlContents.Controls.Add(cpuCtrl);
}

publicvoid LoadRAM()
{
    RAMControl ramCtrl = newRAMControl();
    ramCtrl.mainForm = this;
pnlContents.Controls.Clear();
    ramCtrl.Dock = DockStyle.Fill;
pnlContents.Controls.Add(ramCtrl);
}

publicvoid LoadMotherboard()
{
    MotherboardControl mCtrl = newMotherboardControl();
    mCtrl.mainForm = this;
pnlContents.Controls.Clear();
    mCtrl.Dock = DockStyle.Fill;
pnlContents.Controls.Add(mCtrl);
}

publicvoid LoadGraphics()
{
    GraphicsControl graphics = newGraphicsControl();
    graphics.mainForm = this;
pnlContents.Controls.Clear();
    graphics.Dock = DockStyle.Fill;
pnlContents.Controls.Add(graphics);
}

publicvoid LoadHD()
{
    HDControl hdCtrl = newHDControl();
    hdCtrl.mainForm = this;
pnlContents.Controls.Clear();
    hdCtrl.Dock = DockStyle.Fill;
pnlContents.Controls.Add(hdCtrl);
}

publicvoid LoadOpticalDrives()
{
    OpticalDrivesControl opCtrl = newOpticalDrivesControl();
    opCtrl.mainForm = this;
pnlContents.Controls.Clear();
    opCtrl.Dock = DockStyle.Fill;
pnlContents.Controls.Add(opCtrl);
}

publicvoid LoadAudio()
{
    AudioControl audioCtrl = newAudioControl();
    audioCtrl.mainForm = this;
pnlContents.Controls.Clear();
    audioCtrl.Dock = DockStyle.Fill;
pnlContents.Controls.Add(audioCtrl);
}

```

```

    }

    public void LoadPeripherals()
    {
        PeripheralsControl peripherals = new PeripheralsControl();
        peripherals.mainForm = this;
        pnlContents.Controls.Clear();
        peripherals.Dock = DockStyle.Fill;
        pnlContents.Controls.Add(peripherals);
    }

    public void LoadNetwork()
    {
        NetworkControl networkCtrl = new NetworkControl();
        networkCtrl.mainForm = this;
        pnlContents.Controls.Clear();
        networkCtrl.Dock = DockStyle.Fill;
        pnlContents.Controls.Add(networkCtrl);
    }

    public void LoadPerformance()
    {
        PerformanceControl performanceCtrl = new PerformanceControl();
        performanceCtrl.mainForm = this;
        pnlContents.Controls.Clear();
        performanceCtrl.Dock = DockStyle.Fill;
        pnlContents.Controls.Add(performanceCtrl);
    }

    private void exitToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }

    private void checkForUpdateToolStripMenuItem_Click(object sender, EventArgs e)
    {
        string version = Application.ProductVersion;
        System.Diagnostics.Process.Start("http://www.autospec.com/version.php?v=" + version);
    }

    private void performanceToolStripMenuItem_Click(object sender, EventArgs e)
    {
        LoadPerformance();
    }

    private void lnkPerformance_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
    {
        LoadPerformance();
    }

    private void saveAsTextFileToolStripMenuItem_Click(object sender, EventArgs e)
    {
        string Text2Save = pnlContents.Controls[0].Name == "SummaryControl" ? summary :
        textToSave;
        string filename = pnlContents.Controls[0].Name;
        saveFileDialog1.InitialDirectory =
        Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
    }

```



```

        saveFileDialog1.FileName = saveFileDialog1.InitialDirectory + "\\\" + filename
+ ".txt";
if (saveFileDialog1.ShowDialog() != DialogResult.Cancel)
{
    System.IO.File.WriteAllText(saveFileDialog1.FileName, Text2Save);
    System.Diagnostics.Process.Start(saveFileDialog1.FileName);
}

privatevoid printToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (printDialog1.ShowDialog() != DialogResult.Cancel)
    printDocument1.Print();
}

privatevoid printDocument1_PrintPage(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
    string Text2Save = pnlContents.Controls[0].Name == "SummaryControl" ? summary :
textToSave;
    e.Graphics.DrawString(Text2Save, newFont("Times New Roamn", 12, FontStyle.Regular),
Brushes.Black, 10, 10);
}
}

```

Appendix E

Graphics control source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;

namespace AutoSpec
{
    public partial class GraphicsControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public GraphicsControl()
        {
            InitializeComponent();
        }

        private void GraphicsControl_Load(object sender, EventArgs e)
        {
            ManagementClass mgmt = new ManagementClass("Win32_DisplayConfiguration");
            //create our ManagementObjectCollection to get the attributes with
            ManagementObjectCollection objCol = mgmt.GetInstances();
            string gateway = String.Empty;
            //loop through all the objects we find
            foreach (ManagementObject obj in objCol)
            {
                //Displays the graphics details of the system
                rtfGraphicsDetails.Text += "Name: " + obj["DeviceName"].ToString() +
                "\nMonitor Width: " + Screen.PrimaryScreen.WorkingArea.Width +
                "\nMonitor Height: " + Screen.PrimaryScreen.WorkingArea.Height + " \nMonitor Frequency: " +
                obj["DisplayFrequency"].ToString() + "Hz\n" +
                "Work Resolution: " + Screen.PrimaryScreen.WorkingArea.Width + " x " +
                Screen.PrimaryScreen.WorkingArea.Height + " pixels\n";
            }
            mainForm.textToSave = "GRAPHICS\n" + rtfGraphicsDetails.Text;
        }

        private void GraphicsControl_Resize(object sender, EventArgs e)
        {
            //Makes the graphics details control to resize its size as the main control resizes
            rtfGraphicsDetails.Width = this.Width - 24;
            rtfGraphicsDetails.Height = this.Height - 40;
        }
    }
}
```

Appendix F

HardwareInfo class source code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Management;
using System.IO;
using System.Collections;
using System.Windows.Forms;

namespace AutoSpec
{
    public static class HardwareInfo
    {
        ///<summary>
        /// Retrieving Processor Id.
        ///</summary>
        ///<returns></returns>
        ///

        public static String GetProcessorId()
        {

            ManagementClass mc = new ManagementClass("win32_processor");
            ManagementObjectCollection moc = mc.GetInstances();
            String Id = String.Empty;
            foreach (ManagementObject mo in moc)
            {

                Id = mo.Properties["processorID"].Value.ToString();
            }
            break;
            return Id;
        }

        ///<summary>
        /// Retrieving HDD Serial No.
        ///</summary>
        ///<returns></returns>
        public static String GetHDDSerialNo()
        {
            ManagementClass mangnmt = new ManagementClass("Win32_LogicalDisk");
            ManagementObjectCollection mcol = mangnmt.GetInstances();
            string result = "";
            foreach (ManagementObject strt in mcol)
            {
                result += Convert.ToString(strt["VolumeSerialNumber"]);
            }
            return result;
        }

        ///<summary>
        /// Retrieving System MAC Address.
        ///</summary>
        ///<returns></returns>
        public static string GetMACAddress()
        {
```

```

ManagementClass mc = newManagementClass("Win32_NetworkAdapterConfiguration");
ManagementObjectCollection moc = mc.GetInstances();
string MACAddress = String.Empty;
foreach (ManagementObject mo in moc)
{
    if (MACAddress == String.Empty)
    {
        if ((bool)mo["IPEnabled"] == true) MACAddress = mo["MacAddress"].ToString();
    }
    mo.Dispose();
}

MACAddress = MACAddress.Replace(":", "");
return MACAddress;
}
///

```

```

        }

return "Product: Unknown";

    }
    ///<summary>
    /// Retrieving CD-DVD Drive Path.
    ///</summary>
    ///<returns></returns>
    public static string GetCdRomDrive()
    {

        ManagementObjectSearcher searcher = new ManagementObjectSearcher("root\\CIMV2", "SELECT *
        FROM Win32_CDROMDrive");

        foreach (ManagementObject wmi in searcher.Get())
        {
            try
            {
                return wmi.GetPropertyValue("Drive").ToString();
            }

            catch { }
        }

        return "CD ROM Drive Letter: Unknown";
    }

    ///<summary>
    /// Retrieving BIOS Maker.
    ///</summary>
    ///<returns></returns>
    public static string GetBIOSmaker()
    {

        ManagementObjectSearcher searcher = new ManagementObjectSearcher("root\\CIMV2", "SELECT *
        FROM Win32_BIOS");

        foreach (ManagementObject wmi in searcher.Get())
        {
            try
            {
                return wmi.GetPropertyValue("Manufacturer").ToString();
            }

            catch { }
        }

        return "BIOS Maker: Unknown";
    }

    ///<summary>
    /// Retrieving BIOS Serial No.
    ///</summary>

```

```

///<returns></returns>
publicstaticstring GetBIOSserNo()
{

ManagementObjectSearcher searcher = newManagementObjectSearcher("root\\CIMV2", "SELECT *
FROM Win32_BIOS");

foreach (ManagementObject wmi in searcher.Get())
{
try
{
return wmi.GetPropertyValue("SerialNumber").ToString();

}

catch { }

}

return"BIOS Serial Number: Unknown";

}
///<summary>
/// Retrieving BIOS Caption.
///</summary>
///<returns></returns>
publicstaticstring GetBIOSCaption()
{

ManagementObjectSearcher searcher = newManagementObjectSearcher("root\\CIMV2", "SELECT *
FROM Win32_BIOS");

foreach (ManagementObject wmi in searcher.Get())
{
try
{
return wmi.GetPropertyValue("Caption").ToString();

}

catch { }

}

return"BIOS Caption: Unknown";

}
///<summary>
/// Retrieving System Account Name.
///</summary>
///<returns></returns>
publicstaticstring GetAccountName()
{

ManagementObjectSearcher searcher = newManagementObjectSearcher("root\\CIMV2", "SELECT *
FROM Win32_UserAccount");

foreach (ManagementObject wmi in searcher.Get())
{
try
{

```

```

return wmi.GetPropertyValue("Name").ToString();
    }
catch { }
    }
return "User Account Name: Unknown";

    }
///

```

```

///<returns>CPU Manufacturer</returns>
publicstaticstring GetCPUManufacturer()
{
    string cpuMan = String.Empty;
    //create an instance of the Managemnet class with the
    //Win32_Processor class
    ManagementClass mgmt = newManagementClass("Win32_Processor");
    //create a ManagementObjectCollection to loop through
    ManagementObjectCollection objCol = mgmt.GetInstances();
    //start our loop for all processors found
    foreach (ManagementObject obj in objCol)
    {
        if (cpuMan == String.Empty)
        {
            // only return manufacturer from first CPU
            cpuMan = obj.Properties["Manufacturer"].Value.ToString();
        }
    }
    return cpuMan;
}

///<summary>
/// method to retrieve the CPU's current
/// clock speed using the WMI class
///</summary>
///<returns>Clock speed</returns>
publicstaticint GetCPUCurrentClockSpeed()
{
    int cpuClockSpeed = 0;
    //create an instance of the Managemnet class with the
    //Win32_Processor class
    ManagementClass mgmt = newManagementClass("Win32_Processor");
    //create a ManagementObjectCollection to loop through
    ManagementObjectCollection objCol = mgmt.GetInstances();
    //start our loop for all processors found
    foreach (ManagementObject obj in objCol)
    {
        if (cpuClockSpeed == 0)
        {
            // only return cpuStatus from first CPU
            cpuClockSpeed = Convert.ToInt32(obj.Properties["CurrentClockSpeed"].Value.ToString());
        }
    }
    //return the status
    return cpuClockSpeed;
}

///<summary>
/// method to retrieve the network adapters
/// default IP gateway using WMI
///</summary>
///<returns>adapters default IP gateway</returns>
publicstaticstring GetDefaultIPGateway()
{
    //create out management class object using the
    //Win32_NetworkAdapterConfiguration class to get the attributes
    //of the network adapter
    ManagementClass mgmt = newManagementClass("Win32_NetworkAdapterConfiguration");
    //create our ManagementObjectCollection to get the attributes with

```



```

ManagementObjectCollection objCol = mgmt.GetInstances();
string gateway = String.Empty;
//loop through all the objects we find
foreach (ManagementObject obj in objCol)
{
    if (gateway == String.Empty) // only return MAC Address from first card
    {
        //grab the value from the first network adapter we find
        //you can change the string to an array and get all
        //network adapters found as well
        //check to see if the adapter's IPEnabled
        //equals true
        if ((bool)obj["IPEnabled"] == true)
        {
            gateway = obj["DefaultIPGateway"].ToString();
        }
    }
    //dispose of our object
    obj.Dispose();
}
//replace the ":" with an empty space, this could also
//be removed if you wish
gateway = gateway.Replace(":", "");
//return the mac address
return gateway;
}
///<summary>
/// Retrieve CPU Speed.
///</summary>
///<returns></returns>

public static double? GetCpuSpeedInGHz()
{
    double? GHz = null;
    using (ManagementClass mc = new ManagementClass("Win32_Processor"))
    {
        foreach (ManagementObject mo in mc.GetInstances())
        {
            GHz = 0.001 * (UInt32)mo.Properties["CurrentClockSpeed"].Value;
            break;
        }
    }
    return GHz;
}
///<summary>
/// Converts size of memory location to string
///</summary>
///<param name="size"></param>
///<returns></returns>
public static string SizeToString(long size)
{
    string strSize = "";
    if (size < 1024)
        strSize = size + " bytes";
    elseif (size > 1024 && size < (1024 * 1024))
        strSize = Math.Round(size / 1024.0, 2) + " KB";
    elseif (size > (1024 * 1024) && size < (1024 * 1024 * 1024))
        strSize = Math.Round(size / (1024.0 * 1024.0), 2) + " MB";
}

```

```

elseif (size > (1024 * 1024 * 1024) && size < (1024.0 * 1024 * 1024 * 1024))
    strSize = Math.Round(size / (1024.0 * 1024.0 * 1024.0), 2) + " GB";
else
    strSize = Math.Round(size / (1024.0 * 1024.0 * 1024.0 * 1024.0), 2) + " TB";
return strSize;
}

///

```

```

///<summary>
/// Retrieving Audio Info.
///</summary>
///<returns></returns>
publicstaticstring GetAudioInfo()
{
    string AudioDetails = "";
    ManagementObjectSearcher mo = newManagementObjectSearcher("select * from
Win32_SoundDevice");

    foreach (ManagementObject soundDevice in mo.Get())
    {
        //String deviceId = soundDevice.GetPropertyValue("DeviceId").ToString();
        String name = soundDevice.GetPropertyValue("Name").ToString();

        AudioDetails += name + "\n\n";
    }
    break;
    return AudioDetails;
}
///<summary>
/// Gets basic monitor information
///</summary>
///<returns>graphicInfo</returns>
publicstaticstring GetGraphicsBasicInfo()
{
    string graphicInfo = "";
    ManagementClass mgmt = newManagementClass("Win32_DisplayConfiguration");
    //create our ManagementObjectCollection to get the attributes with
    ManagementObjectCollection objCol = mgmt.GetInstances();
    string gateway = String.Empty;
    //loop through all the objects we find
    foreach (ManagementObject obj in objCol)
    {
        graphicInfo += obj["DeviceName"].ToString() + " (" +
Screen.PrimaryScreen.WorkingArea.Width +
" x " + Screen.PrimaryScreen.WorkingArea.Height + " @ " +
obj["DisplayFrequency"].ToString() + "Hz)";

    }
    return graphicInfo;
}
}
}

```

Appendix G

Hard Drive control source code

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;
using System.Collections;

namespace AutoSpec
{
    public partial class HDControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public HDControl()
        {
            InitializeComponent();

            private void HDControl_Load(object sender, EventArgs e)
            {
                ManagementObjectSearcher searcher = new ManagementObjectSearcher("SELECT * FROM Win32_DiskDrive");

                foreach (ManagementObject hd in searcher.Get())
                {
                    try
                    {
                        //displays the hard disk details
                        rtfHDdetails.Text += "Media Type: " +
                            hd.GetPropertyValue("MediaType").ToString() + "\nSerial Number: " +
                            hd.GetPropertyValue("SerialNumber").ToString() + "\nName: " +
                            hd.GetPropertyValue("Name").ToString() + "\nTotal Cylinders: " +
                            hd.GetPropertyValue("TotalCylinders").ToString() + "\nTotal Size: " +
                            HardwareInfo.GetTotalLocalDiskSize() + "\n" +
                            "Total Heads: " + hd.GetPropertyValue("TotalHeads").ToString() + "\nTotal Sectors: " +
                            hd.GetPropertyValue("TotalSectors").ToString() + "\nTotal Tracks:" +
                            hd.GetPropertyValue("TotalTracks").ToString() + "\nTracks Per Cylinder: " +
                            hd.GetPropertyValue("TracksPerCylinder").ToString() + "\n\n" ;
                    }
                    catch (Exception ex) { }
                }
                mainForm.textToSave = "HARD DRIVES\n" + rtfHDdetails.Text;
            }

            private void HDControl_Resize(object sender, EventArgs e)
            {
                rtfHDdetails.Width = this.Width - 24;
                rtfHDdetails.Height = this.Height - 40;
            }
        }
    }
}
class HardDrive
{

```

```

privatestring model = null;
privatestring type = null;
privatestring serialNo = null;
ArrayList hdCollection = newArrayList();
publicstring Model
{
    get { return model; }
    set { model = value; }
}
publicstring Type
{
    get { return type; }
    set { type = value; }
}
publicstring SerialNo
{
    get { return serialNo; }
    set { serialNo = value; }
}

public HardDrive()
{

ManagementObjectSearcher searcher = newManagementObjectSearcher("SELECT * FROM
Win32_DiskDrive");

/*foreach (ManagementObject wmi_HD in searcher.Get())
{
    HardDrive hd = new HardDrive();
    hd.Model = wmi_HD["Model"].ToString();
    hd.Type = wmi_HD["InterfaceType"].ToString();
hdCollection.Add(hd);
}*/

searcher = newManagementObjectSearcher("SELECT * FROM Win32_PhysicalMedia");

int i = 0;
foreach (ManagementObject wmi_HD in searcher.Get())
{
    // get the hard drive from collection
    // using index
    HardDrive hd = (HardDrive)hdCollection[i];

    // get the hardware serial no.
    if (wmi_HD["SerialNumber"] == null)
        hd.SerialNo = "None";
    else
        hd.SerialNo = wmi_HD["SerialNumber"].ToString();

        ++i;
    }
}
}

```

Appendix H

Motherboard Control source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;

namespace AutoSpec
{
    public partial class MotherboardControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public MotherboardControl()
        {
            InitializeComponent();
        }

        public string getMotherboardDetails()
        {
            SelectQuery Sq = new SelectQuery("Win32_MotherboardDevice");
            ManagementObjectSearcher objOSDetails = new ManagementObjectSearcher(Sq);
            ManagementObjectCollection osDetailsCollection = objOSDetails.Get();
            StringBuilder sb = new StringBuilder();
            foreach (ManagementObject mo in osDetailsCollection)
            {
                //this portion of the program formats the motherboard details of the system
                sb.AppendLine(string.Format("Caption: {0}", (string)mo["Caption"]));
                sb.AppendLine(string.Format("Availability: {0}", mo["Availability"].ToString()));
                sb.AppendLine(string.Format("InstallDate: {0}",
                    Convert.ToDateTime(mo["InstallDate"]).ToString()));
                sb.AppendLine(string.Format("CreationClassName : {0}", (string)mo["CreationClassName"]));
                sb.AppendLine(string.Format("Description: {0}", (string)mo["Description"]));
                sb.AppendLine(string.Format("DeviceID : {0}", (string)mo["DeviceID"]));
                sb.AppendLine(string.Format("ErrorCleared: {0}", (string)mo["ErrorCleared"]));
                sb.AppendLine(string.Format("ErrorDescription : {0}", (string)mo["ErrorDescription"]));
                sb.AppendLine(string.Format("PrimaryBusType : {0}", (string)mo["PrimaryBusType"]));
                sb.AppendLine(string.Format("RevisionNumber : {0}", (string)mo["RevisionNumber"]));
                sb.AppendLine(string.Format("LastErrorCode : {0}", (string)mo["LastErrorCode"]));
                sb.AppendLine(string.Format("Name : {0}", (string)mo["Name"]));
                sb.AppendLine(string.Format("SecondaryBusType : {0}", (string)mo["SecondaryBusType"]));
                sb.AppendLine(string.Format("PNPDeviceID: {0}", (string)mo["PNPDeviceID"]));
                sb.AppendLine(string.Format("PowerManagementSupported : {0}",
                    mo["PowerManagementSupported"].ToString()));
                sb.AppendLine(string.Format("Status : {0}", (string)mo["Status"]));
                sb.AppendLine(string.Format("SystemCreationClassName : {0}",
                    (string)mo["SystemCreationClassName"]));
                sb.AppendLine(string.Format("SystemName: {0}", (string)mo["SystemName"]));
            }
            return sb.ToString();
        }

        private void MotherboardControl_Load(object sender, EventArgs e)
```

```

        {
            rtfDetails.Text = getMotherboardDetails(); //displays the motherboard details
            rtfDetails.Text += "\n\nBIOS:\n\tMaker: " + HardwareInfo.GetBIOSmaker();
            rtfDetails.Text += "\n\tCaption: " + HardwareInfo.GetBIOScaption()
+ "\n\tSerial no: " + HardwareInfo.GetBIOSserNo();
            MainForm.textToSave = "MOTHERBOARD\n" + rtfDetails.Text;
        }

private void MotherboardControl_Resize(object sender, EventArgs e)
    {
        rtfDetails.Width = this.Width - 24;
        rtfDetails.Height = this.Height - 40;
    }
}
}

```

Appendix I

Network Control source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;

namespace AutoSpec
{
    public partial class NetworkControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public NetworkControl()
        {
            InitializeComponent();
        }

        private void NetworkControl_Load(object sender, EventArgs e)
        {
            rtfNetworkDetails.Text = !AS.HasConnection() ? "You are not connected to the internet\n" : "You are connected to the internet\n";
            //create out management class object using the
            //Win32_NetworkAdapterConfiguration class to get the attributes
            //of the network adapter
            ManagementClass mgmt = new ManagementClass("Win32_NetworkAdapterConfiguration");
            //create our ManagementObjectCollection to get the attributes with
            ManagementObjectCollection objCol = mgmt.GetInstances();
            string gateway = String.Empty;
            //loop through all the objects we find
            foreach (ManagementObject obj in objCol)
            {
                //rtfNetworkDetails.Text += obj["Description"].ToString() + "\n";
            }

            ManagementObjectSearcher mo = new ManagementObjectSearcher("select * from Win32_ComputerSystem");

            foreach (ManagementObject osDesc in mo.Get())
            {
                //displays the network details
                rtfNetworkDetails.Text += "Username: " +
                osDesc.GetPropertyValue("UserName").ToString() + "\n" +
                "Workgroup: " + osDesc.GetPropertyValue("Workgroup").ToString() + "\n" +
                "DNS Host name: " + osDesc.GetPropertyValue("DNSHostName").ToString() + "\n" +
                "Domain: " + osDesc.GetPropertyValue("Domain").ToString() + "\n" +
                "Description: " + osDesc.GetPropertyValue("Description").ToString() + "\n\n";
            }

            rtfNetworkDetails.Text += "\n\nNetwork Adapters:\n\t";
            mgmt = new ManagementClass("Win32_NetworkAdapter");
            //create our ManagementObjectCollection to get the attributes with
```



```

ManagementObjectCollection objCol2 = mgmt.GetInstances();
//loop through all the objects we find
foreach (ManagementObject obj in objCol2)
{
    try
    {
        rtfNetworkDetails.Text += "Name: " + obj["Name"].ToString() + "\n\t";
        rtfNetworkDetails.Text += "Device ID: " + obj["DeviceID"].ToString()
+ "\n\t";
        //rtfNetworkDetails.Text += "Service name: " + obj["ServiceName"].ToString() + "\n\t";
        rtfNetworkDetails.Text += "Manufacturer: " +
obj["Manufacturer"].ToString() + "\n\n\t";
    }
    catch (Exception ex) { }

    }
    MainForm.textToSave = "NETWORK\n" + rtfNetworkDetails.Text;
}

private void NetworkControl_Resize(object sender, EventArgs e)
{
    rtfNetworkDetails.Width = this.Width - 24;
    rtfNetworkDetails.Height = this.Height - 40;
}

}
}

```

Appendix J

Optical drives control source code

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace AutoSpec
{
    public partial class OpticalDrivesControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public OpticalDrivesControl()
        {
            InitializeComponent();
        }
        private void OpticalDrivesControl_Load(object sender, EventArgs e)
        {
            bool _empty_drives = true;
            DriveInfo[] ListDrives = DriveInfo.GetDrives(); //retrieves all the drives in the system
            if (ListDrives.Length == 0)
            {
                rtfOpticalDrivesDetails.Text = "No optical disk drives detected";
                return;
            }
            int d = 0;
            foreach (DriveInfo Drive in ListDrives)
            {
                ++d;
                if (Drive.DriveType == DriveType.Removable && Drive.IsReady) //seperates removable drives
                    from other drives
                {
                    //displays the optical drives
                    rtfOpticalDrivesDetails.Text += "Drive " + d + ": " + Drive.Name +
                    (Drive.IsReady ? "\n\tLabel: " + Drive.VolumeLabel + "\n" : "\n");
                    rtfOpticalDrivesDetails.Text += "\tTotal space: " +
                    HardwareInfo.SizeToString(Drive.TotalSize) + "\n\tUsed Space: " +
                    HardwareInfo.SizeToString(Drive.TotalSize - Drive.TotalFreeSpace) + "\n";
                    rtfOpticalDrivesDetails.Text += "\tFree space: " +
                    HardwareInfo.SizeToString(Drive.TotalFreeSpace) + "\n\n";
                    _empty_drives = false;
                }
            }
            if(_empty_drives)
                rtfOpticalDrivesDetails.Text = "No optical disk drives detected";
            mainForm.textToSave = "OPTICAL DRIVES\n" + rtfOpticalDrivesDetails.Text;
        }
        private void OpticalDrivesControl_Resize(object sender, EventArgs e)
        {
            rtfOpticalDrivesDetails.Width = this.Width - 24;
            rtfOpticalDrivesDetails.Height = this.Height - 40;
        }
    }
}

```

Appendix K

Operating system control source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;

namespace AutoSpec
{
    public partial class OSControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public OSControl()
        {
            InitializeComponent();

            private void tmrOS_Tick(object sender, EventArgs e)
            {
            }

            private void OSControl_Load(object sender, EventArgs e)
            {
                string osBits = Environment.Is64BitOperatingSystem ? " 64-bit" : " 32-bit";
                Microsoft.VisualBasic.Devices.ComputerInfo ci =
                new Microsoft.VisualBasic.Devices.ComputerInfo();
                rtfOSDetails.Text = ci.OSFullName + "" + osBits + "\nVersion: " +
                ci.OSVersion + "\n";
                ManagementObjectSearcher mo = new ManagementObjectSearcher("select * from
                Win32_OperatingSystem");

                foreach (ManagementObject osDesc in mo.Get())
                {
                    //Displays the operating systems details
                    rtfOSDetails.Text += "Manufacturer: " +
                    osDesc.GetPropertyValue("Manufacturer").ToString() + "\n" +
                    "Code Set: " + osDesc.GetPropertyValue("CodeSet").ToString() + "\n" +
                    "Computer name: " + osDesc.GetPropertyValue("CSName").ToString() + "\n" +
                    "Registered user: " + osDesc.GetPropertyValue("RegisteredUser").ToString() + "\n" +
                    "Serial number: " + osDesc.GetPropertyValue("SerialNumber").ToString() + "\n" +
                    "System Drive: " + osDesc.GetPropertyValue("SystemDrive").ToString() + "\n" +
                    "Installation Date: " + FormatDate(osDesc.GetPropertyValue("InstallDate").ToString()) +
                    "\n";
                }
                mainForm.textToSave = "OPERATING SYSTEM\n" + rtfOSDetails.Text;
            }

            private void OSControl_Resize(object sender, EventArgs e)
            {
            }
        }
    }
}
```

```

//Makes the os details control to resize its size as the main control resizes
    rtfOSDetails.Width = this.Width - 24;
    rtfOSDetails.Height = this.Height - 40;
}

privatestring FormatDate(string strDate) //this method formats the install date
{
    string y = strDate.Substring(0, 4);
    string m = strDate.Substring(4, 2);
    string d = strDate.Substring(6, 2);
    return d + "/" + m + "/" + y;
}
}
}

```

Appendix L

Performance control source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Collections;
using System.Windows.Forms;
using System.Diagnostics;
using System.Management;

namespace AutoSpec
{
    public partial class PerformanceControl : UserControl
    {
        ArrayList possibleSolutions = new ArrayList();
        public frmMain mainForm = new frmMain();
        public PerformanceControl()
        {
            InitializeComponent();

            private double PercentageHDUsage()
            {
                long[] hdSize = HardwareInfo.GetTotalLocalDiskSizes(); //gets logical disk sizes
                return Math.Round((double)hdSize[1] / hdSize[0], 2) * 100; //percentate cpu usage
            }

            private bool IsSmallRamSize()
            {
                ManagementScope oMs = new ManagementScope();
                ObjectQuery oQuery = new ObjectQuery("SELECT * FROM Win32_PhysicalMemory");
                ManagementObjectSearcher oSearcher = new ManagementObjectSearcher(oMs, oQuery);
                ManagementObjectCollection oCollection = oSearcher.Get();

                long MemSize = 0;
                long speed = 0;
                long mCap = 0;
                string channel = "";
                // In case more than one Memory sticks are installed
                foreach (ManagementObject obj in oCollection)
                {
                    mCap = Convert.ToInt64(obj["Capacity"]); //memory capacity
                    MemSize += mCap; //adds up memory capacity
                }
                MemSize = (MemSize / 1024) / 1024; //converts from byte to mega byte
                return (MemSize / 1024) < 2;
            }

            //Gets all the running applications
            private string[] getRunningApps()
            {
                ArrayList runningApps = new ArrayList();
            }
        }
    }
}
```

```

Process[] processes = Process.GetProcesses();
foreach (Process process in processes)
{
    if (process.MainWindowTitle.Trim().Length > 0) //helps to seperate open apps from
    background processes
    runningApps.Add(process.MainWindowTitle);
}
return (string[])runningApps.ToArray(typeof(string)); //returns an array of running apps
}

privatevoid PerformanceControl_Resize(object sender, EventArgs e)
{
    rtfPerformanceDetails.Width = this.Width - 24;
    rtfPerformanceDetails.Height = this.Height - 40;
}

privatevoid tmrPerformance_Tick(object sender, EventArgs e)
{
    /*long ramSize = Convert.ToInt64(HardwareInfo.GetPhysicalMemory(true));
    PerformanceCounter performanceCounter = new PerformanceCounter("Processor",
    "% Processor Time", "_Total");//"Process", "Private Bytes", "Explorer");
    lblPerformance.Text = performanceCounter.CounterName + ": " +
    performanceCounter.RawValue;
    label1.Text = GetAvailableRam()/1024 + "GB" + "/" +
    HardwareInfo.GetPhysicalMemory();
    //lblProcessMaxSpeed.Text = HardwareInfo.GetCPUCurrentClockSpeed() + "*/
}

privatevoid tmrPerformance2_Tick(object sender, EventArgs e)
{
    long ramSize = Convert.ToInt64(HardwareInfo.GetPhysicalMemory(true));
    possibleSolutions = newArrayList();
    string runningApps = "";
    string runningProcesses = "";
    rtfPerformanceDetails.Text = "CAUSES OF CPU LOW PERFORMANCE\n\tToo many
background processes";
    possibleSolutions.Add("Use the Task Manager to kill unnecessary background processes");
    if (PercentageHDDUsage() > 95) //checks for hard disk 95% usage
    {
        rtfPerformanceDetails.Text += "\n\tLow available space in hard disk";
        possibleSolutions.Add("Free up your hard disk to boost performance");
    }
    if (TotalRunningApps() > 20) //checks whether running applications are greater than 20
    {
        rtfPerformanceDetails.Text += "\n\tToo many open applications running";
        possibleSolutions.Add("Close some of the applications running");
    }
    if (IsSmallRamSize()) //checks for small size RAM
    {
        rtfPerformanceDetails.Text += "\n\tLow RAM size";
        possibleSolutions.Add("Upgrade your RAM size to at least 2GB");
    }
    if (possibleSolutions.Count > 0)
    {
        rtfPerformanceDetails.Text += "\nSOLUTIONS\n\t";
        for (int i = 0; i < possibleSolutions.Count; i++)
        {

```

```

                rtfPerformanceDetails.Text += possibleSolutions[i].ToString() +
"\n\t";
            }
        }
        rtfPerformanceDetails.Text += "\n\nRUNNING PROGRAMS:\n\t";
        Process[] processes = Process.GetProcesses();
        foreach (Process process in processes)
        {
            if (process.MainWindowTitle.Trim().Length > 0)
                runningApps += process.MainWindowTitle + "\n\t"; //collects running apps
            else
                runningProcesses += process.ProcessName + "\n\t"; //collects running background processes
        }
        rtfPerformanceDetails.Text += runningApps + "\nBACKGROUND PROCESSES\n\t" +
runningProcesses; //prints the running programs
        mainForm.textToSave = "PERFORMANCE\n" + rtfPerformanceDetails.Text;
        tmrPerformance2.Stop();
        tmrPerformance2.Interval = 30000;
        tmrPerformance2.Start();
    }

    public long GetAvailableRam()
    {
        PerformanceCounter ramCounter = new PerformanceCounter("Memory", "Available MBytes");
        return ramCounter.RawValue;
    }

    private int TotalRunningApps() //returns total running apps
    {
        ArrayList runningApps = new ArrayList();
        Process[] processes = Process.GetProcesses();
        foreach (Process process in processes)
        {
            if (process.MainWindowTitle.Trim().Length > 0)
                runningApps.Add(process.MainWindowTitle);
        }
        return runningApps.Count;
    }
}

```

Appendix M

Peripherals control source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;

namespace AutoSpec
{
    public partial class PeripheralsControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public PeripheralsControl()
        {
            InitializeComponent();

            private void PeripheralsControl_Load(object sender, EventArgs e)
            {
                ManagementObjectSearcher mo = new ManagementObjectSearcher("select * from Win32_PointingDevice");

                foreach (ManagementObject keybdDevice in mo.Get())
                {
                    String deviceId = keybdDevice.GetPropertyValue("DeviceID").ToString(); //gets keyboard ID
                    String name = keybdDevice.GetPropertyValue("Name").ToString(); //gets keyboard name

                    rtfPeripheralDetails.Text += "Name: " + name + "\nDevice ID" + deviceId +
                    "\nManufacturer: " +
                    keybdDevice.GetPropertyValue("Manufacturer") + "\nDevice Type: Keyboard\nDevice
                    Interface: " +
                    GetInterfaceType(Convert.ToInt32(keybdDevice.GetPropertyValue("DeviceInterface"))) +
                    "\n\n";
                }

                rtfPeripheralDetails.Text += "\nPrinters\n\t";
                mo = new ManagementObjectSearcher("select * from Win32_Printer");

                foreach (ManagementObject printerDevice in mo.Get())
                {
                    String deviceId = printerDevice.GetPropertyValue("DeviceID").ToString(); //gets printer
                    ID
                    String name = printerDevice.GetPropertyValue("Name").ToString(); //gets printer name
                    //displays the printer details
                    rtfPeripheralDetails.Text += "Name: " + name + "\n\tDevice ID" + deviceId
                    + "\n\tDriver Name: " +
                    printerDevice.GetPropertyValue("DriverName") + "\n\tPort Name: " +
                    printerDevice.GetPropertyValue("PortName") + "\n\n\t";
                }
                mainForm.textToSave = "PERIPHERALS\n" + rtfPeripheralDetails.Text;
            }
        }
    }
}
```



```

//helps to get the keyboard interface type
private string GetInterfaceType(int t)
{
    if (t == 1)
        return "Other";

    if (t == 2)
        return "Unknown";

    if (t == 3)
        return "Serial";
    if (t == 4)
        return "PS/2";
    if (t == 5)
        return "Infrared";
    if (t == 6)
        return "HP-HIL";
    if (t == 7)
        return "Bus Mouse";
    if (t == 8)
        return "ADB (Apple Desktop Bus)";
    if (t == 160)
        return "Bus Mouse DB-9";
    if (t == 161)
        return "Bus Mouse Micro-DIN";
    if (t == 162)
        return "USB";
    else return "Unknown";
}

private void PeripheralsControl_Resize(object sender, EventArgs e)
{
    {
        rtfPeripheralDetails.Width = this.Width - 24;
        rtfPeripheralDetails.Height = this.Height - 40;
    }
}
}

```

Appendix N

RAM control source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;

namespace AutoSpec
{
    public partial class RAMControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public RAMControl()
        {
            InitializeComponent();

            private void RAMControl_Load(object sender, EventArgs e)
            {
                ManagementObjectSearcher mo = new ManagementObjectSearcher("select * from Win32_PhysicalMemory");

                foreach (ManagementObject ram in mo.Get())
                {
                    String manu = ram.GetPropertyValue("Manufacturer").ToString(); //gets the manufacturer of a RAM
                    String name = ram.GetPropertyValue("Name").ToString(); //gets the name of a RAM
                    //displays the RAM details
                    rtfRamDetails.Text += "Name: " + name + "\nManufacturer: " + manu +
                    "\nModel: " + ram.GetPropertyValue("Model") + "\n";
                    rtfRamDetails.Text += "Data Width: " +
                    ram.GetPropertyValue("DataWidth").ToString() + "\nCapacity: " +
                    HardwareInfo.GetPhysicalMemory() +
                    "\nSpeed: " + ram.GetPropertyValue("Speed") + "\n";
                }
                mainForm.textToSave = "RAM\n" + rtfRamDetails.Text;
            }

            private void RAMControl_Resize(object sender, EventArgs e)
            {
                rtfRamDetails.Width = this.Width - 24;
                rtfRamDetails.Height = this.Height - 40;
            }
        }
    }
}
```

Appendix O

Summary control source code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Management;

namespace AutoSpec
{
    public partial class SummaryControl : UserControl
    {
        public frmMain mainForm = new frmMain();
        public SummaryControl()
        {
            InitializeComponent();
        }

        private void picOperatingSystem_Click(object sender, EventArgs e)
        {
            mainForm.LoadOS();
        }

        private void picCPU_Click(object sender, EventArgs e)
        {
            mainForm.LoadCPU();
        }

        private void picRAM_Click(object sender, EventArgs e)
        {
            mainForm.LoadRAM();
        }

        private void picMotherboard_Click(object sender, EventArgs e)
        {
            mainForm.LoadMotherboard();
        }

        private void picGraphics_Click(object sender, EventArgs e)
        {
            mainForm.LoadGraphics();
        }

        private void picHardDrives_Click(object sender, EventArgs e)
        {
            mainForm.LoadHD();
        }

        private void picOpticalDrives_Click(object sender, EventArgs e)
        {
            mainForm.LoadOpticalDrives();
        }
    }
}
```

```

privatevoid picAudio_Click(object sender, EventArgs e)
{
    mainForm.LoadAudio();
}

privatevoid tmrSummary_Tick(object sender, EventArgs e)
{
    string osBits = Environment.Is64BitOperatingSystem ? " 64-bit" : " 32-bit"; //detects
    os bits 32 or 64
    Microsoft.VisualBasic.Devices.ComputerInfo ci =
    newMicrosoft.VisualBasic.Devices.ComputerInfo();
    Microsoft.VisualBasic.Devices.Computer computer =
    newMicrosoft.VisualBasic.Devices.Computer();
    lblOS.Text = ci.OSFullName + osBits; //displays operating system summary
    lblRAM.Text = HardwareInfo.GetPhysicalMemory(); //displays RAM summary
    lblHardDrives.Text = HardwareInfo.GetTotalLocalDiskSize(); //displays hard
    disk summary
    lblMotherboard.Text = HardwareInfo.GetBoardMaker(); //displays motherboard
    summary
    lblAudio.Text = HardwareInfo.GetAudioInfo(); ////displays audio summary
    lblGraphics.Text = HardwareInfo.GetGraphicsBasicInfo(); //displays graphics
    summary
    DriveInfo[] dList = HardwareInfo.GetRemovableDrives();
    string drives = "";
    if (dList.Length > 0)
    {
        foreach (DriveInfo di in dList)
        {
            drives += di.Name + "\n";
        }
        lblOpticalDrives.Text = drives;
    }
    else
        lblOpticalDrives.Text = "No optical disk drives detected";
    mainForm.summary = "SUMMARY\nOperating System: " + lblOS.Text + "\n\nCPU: " +
    lblCPU.Text + "\n\nRAM: " + lblRAM.Text +
    "Motherboard: " + lblMotherboard.Text + "\n\nGraphics: " + lblGraphics.Text + "\n\nHard
    Drives: " + lblHardDrives.Text +
    "\n\nOptical Drives: " + lblOpticalDrives.Text + "\n\nAudio: " + lblAudio.Text;
}

privatevoid SummaryControl_Load(object sender, EventArgs e)
{
    string cpu = "";
    using (ManagementObjectSearcher win32Proc = newManagementObjectSearcher("select * from
    Win32_Processor"),
        win32CompSys = newManagementObjectSearcher("select * from
    Win32_ComputerSystem"),
        win32Memory = newManagementObjectSearcher("select * from
    Win32_PhysicalMemory"))
    {
        foreach (ManagementObject obj in win32Proc.Get())
        {
            string clockSpeed = obj["CurrentClockSpeed"].ToString(); //current clock speed

```

```

string procName = obj["Name"].ToString(); //name of processor
string manufacturer = obj["Manufacturer"].ToString(); //gets the manufacturer of the
processor
string version = obj["Version"].ToString();
cpu += procName; // +"\\n" + manufacturer + "\\n" + version;
    }
    }
    lblCPU.Text = cpu;
}
}
}

```