Executable and resource-predicting application models for interventional X-ray

Korikkar, M.

Award date:
2013

Executable and Resource-Predicting Application Models for interventional X-Ray

Madhura Korikkar
0827592

Master Thesis

Eindhoven University of Technology
Department of Mathematics and Computer Science
System Architecture and Networking
and
Manipal Institute of Technology
Department of Information and Communication Technology
Network Engineering

Graduation Supervisor:
prof.dr. J. J. Lukkien
Chair, System Architecture and Networking
j.j.lukkien@tue.nl

Supervisor at MIT, Manipal:
prof.dr. Radhika M. Pai
radhika.pai@manipal.edu

Tutor:
ir. A. Visser
System Architect
Philips Healthcare, interventional X-ray
alex.visser@philips.com

Abstract

With the advent of improving technology, advanced solutions are provided to healthcare professionals. Through combining human insights and clinical expertise, patient outcomes are improved while lowering the burden on the healthcare system. **Interventional X-Ray** is responsible for the development of X-Ray systems used for diagnosis and interventional treatment of cardiac and vascular diseases. Philips Healthcare is the market leader in this field.

In Cardiovascular minimal invasive interventions, physicians require low-latency X-Ray imaging applications, as their actions must be directly visible on the screen. This requires sufficient performance of the image-processing system while executing a plurality of functions. Because dedicated hardware lacks flexibility, the functionality of such systems is increasingly based on commercial technology like PC hardware, Operating Systems(OS) and state of the art software languages. Therefore, the development of this type of systems is performed in multidisciplinary teams where hardware, mechanics and software come together.

A clear trend within interventional X-Ray applications is a complete integration of diagnostic features within X-Ray interventions, to support earlier feedback in the clinical work-flow. Interventional image processing and other diagnostic applications are combined in one system design and executed simultaneously in various combinations.

A consequence of the sketched trend is that multiple applications are executed in parallel and that the PCs used for these systems have to offer sufficient performance under various use-cases. The latter has to be verified for each X-Ray system that is assembled during production. However, adequate prediction and verification of the components based on modeling beforehand saves time and investment. They can also be given to hardware vendors for use in the selection process.

In addition to this, models can be be used to decide the optimum combination of applications on a specific hardware platform.

The goal of the work is therefore to make models of applications that admit prediction of their execution behaviour on a given platform. These models will be both descriptive

and executable. More precisely, the work includes the following tasks:

- Define the technical context (assumptions, requirements, and existing work) for this prediction and verification
- Define the particular requirements for the workstations within the X-Ray system and relevant parameters
- Collect sufficient measurement data of the resource usage of the applications to develop descriptive models. These models predict the application behaviour sufficiently precise on a given platform
- Develop executable models, based on micro-benchmarks, to stress test PC workstations for their intended use, without the actual use of the medical software application. This stress test shall verify the actual performance for a specific instance of a platform. Under-performing PCs can then be excluded from the actual assembly.

Acknowledgements

This thesis is written as part of the master project that I have performed under the chair of System Architecture and Networking at Eindhoven University of Technology, The Netherlands, in cooperation with Philips Healthcare, Best, The Netherlands.

Upon completion of my master thesis, I would like to extend my heartfelt gratitude to everyone who has contributed to the successful completion of this work. My venture with this project has been extremely motivating and interesting. I have been able to learn and explore and broaden my horizon through this project. Therefore, first and foremost, I would like to express my utmost gratitude to my graduation supervisor, prof.dr. Johan J. Lukkien who gave me the magnificent opportunity of working at Philips Healthcare. He has been a terrific guide and motivator. Discussions and interactions with him have helped me think and explore things on a deeper level. I thank him for his encouragement, guidance, recurring feedbacks and for helping me shape my thesis into what it is today. I would also like to express my deepest and sincere gratitude to my tutor and supervisor at Philips Healthcare, Alex Visser who has been guiding and helping me to think independently and providing his valuable time and feedback. I also thank him for coming up with the idea of Sounding Board meeting at Philips Healthcare. These meetings helped me develop my insight and understand my work better. He inspires me on both, a professional and a personal level. I would also like to thank my mentor at TU/e, dr. Rudolf H. Mak for his valuable suggestions and feedback throughout the execution of the project. He has given new insights to this project during the Sounding Board meetings which helped me improve my work.

I extend my gratitude to prof.dr. Mark van den Brand, the program director for the Manipal-TU/e dual degree program, for being a part of my assessment committee in my project and most importantly for selecting me, along with prof.dr. Manohara M. Pai and prof.dr. Radhika Pai, to do my masters in this prestigious institute. I express my heartfelt gratitude to them for giving me this opportunity. I would also like to thank my supervisor from Manipal Institute of Technology, prof.dr. Radhika Pai for her valuable evaluation and providing me support on a personal level. I would also like to express my sincere respect and gratitude to Pascal Wolkotte, Mike Creemers, Bart Van Rijnsoever for being a part of the Sounding Board committee at Philips Healthcare and providing

useful suggestions, feedback and support in many circumstances. I also thank Philips Healthcare for giving me a new light and thinking in my career and all my colleagues there who helped me with my thesis directly or indirectly.

I would like to thank my friends who stood by me at all times. Finally, I would like to thank my father, Subrahmanya, my mother Lakshmi and my brother, Sumanth for their unconditional encouragement and support. They are the *butter to my bread and the breath to my life.*

Contents

Nomenclature

AVX Advanced Vector Extensions

BIOS Basic Input Output System

CPU Central Processing Unit

DIMM Dual In Line Memory Module

DMI Direct Media Interface

DRAM Dynamic Random Access Memory

ESI Enclosure Services Interface

ICH I/O Controller Hub

IMC Integrated Memory Controller

IP Image Processing

LSI Large Scale Integration

NUMA Non Uniform Memory Access

PCH Platform Controller Hub

PCIe Peripheral Component Interconnect Express

PCM Performance Monitor Counter

QPI Quick Path Interconnect

RUM Resource Usage Measurement

RUMM Resource Usage Modeling Measure

SAS Serial Attached SCSI

SATA Serial Advanced Technology Attachment

SCSI Small Computer System Interface

TDP Thermal Design Power

USB Universal Serial Bus

Chapter 1

Introduction

1.1 Introduction

In today's era of technological boom and revolution, the world has witnessed countless number of wonders in every aspect of life. Medicine and patient health-care is one such field which has saved millions of lives and continues to evolve with the technology. Today it is possible to cure several ailments related to heart, brain and vascular tissues using angiography generated by equipments that work in cord with very efficient and highly powerful PCs. The clinical use of X-Ray imaging for angiography is undergoing a gradual paradigm shift from surgery to intervention. Interventional treatments use X-Ray images during the procedure which demands eye hand co-ordination of the physician and imposes real time requirements on the image processing including low latency. A clear trend in this aspect is therefore the integration of diagnostic and intervention X-Ray imaging to provide earlier feedback in the clinical work-flow and possibly provide guidance to intervention[1]. Allura X-Ray system is one such sophisticated system from Philips Healthcare designed and developed for both diagnostic and interventional purposes. Therefore, it is not only essential, but inevitable for this system to perform with high reliability. This necessitates the use of extensive and elaborate tests during production, assembly and release of the product. In spite of the series of tests conducted, some systems are known to fail due to hardware failures. One goal of this project is to develop executable models that can be run on a system and predict failures and determine its nature. Once these models are created and shipped to the manufacturer of the PC, from the business perspective, it achieves significant cost reduction by avoiding the shipping costs to the manufacturer of the PC.

Allura X-Ray system uses different PCs for different functionalities. Though each of these PCs are of immense use in their distinct ways, combining two or more PCs into one PC is a huge capital saver. Taking this as a motivation, another goal of this project

is to develop descriptive models that can be used to predict the performance of an application, or many combined applications on a PC with known hardware properties. This can be a starting point to combine two PCs into one, or predict the performance of a PC even before the PC is available from the vendor, just based on the architectural properties.

The outcome of this project is a set of descriptive and executable models that predict the performance of a PC. This report gives a detailed description of the design, development of such models and interesting observations and inferences made during the execution of this project.

1.2 Understanding the Allura-Xray System

The development of performance models is based on the sound understanding of the Allura X-Ray system. As a ground step, this section discusses basics related to the Allura X-Ray system.

1.2.1 Biomedical X-Ray imaging

Biomedical X-Ray imaging has proven to be an indispensable component of many medical diagnostic and treatment techniques. X-Ray has been used for biomedical imaging ever since Roentgen X-rayed his wife's hand in 1895 after he discovered it[2]. The speed with which the technology was implemented in medicine is phenomenal by today's standards. An important reason for this rapid acceptance is probably that it was the first time that objective evaluation of an internal organ could be made (without opening the body or surgery) to act in a meaningful way, and the other reason is probably that the technology for making X-Rays and recording the X-Ray images were available because they were already in wide spread use for other purposes.

An X-Ray image is created by radiating the desired part of the human body and capturing the remaining radiation at the opposite side of the body part in an X-Ray detector. Since its discovery, X-Ray has been used to detect bone fractures, gunshot wounds and is now culminating in X-Ray radiography, computed tomography , fluoroscopy and radiotherapy. X-rays can also be used for minimal invasive interventional treatments of heart and brain. Minimal invasive procedures are performed through tiny incisions instead of one large opening used in conventional open surgeries. Because the incisions are small, patients tend to have quicker recovery times and less discomfort than with conventional surgery.

With minimally invasive interventions, cardiologists diagnose and treat a coronary artery

disease, using a catheter inserted into the groin and threaded through the arterial vessel tree to reach the heart. Also, radiologists use these systems to treat aneurysms,thromboses by inserting catheters in the veins. Treatment may be in the form of balloon angioplasty(compressing the plaque against the wall of the vessel), stenting (inserting a small wire tube). A radio opaque contrast fluid which is injected shows the structure of the lumen of the blood vessel through which the catheter passes and pinpoints to blockages or narrowed arteries that need treatment.The contrast fluid blocks the X-Ray and creates contrast between the blood vessels and its surroundings. This principle is used in angiography. Figure 1.1 is a good example of X-Ray angiography image

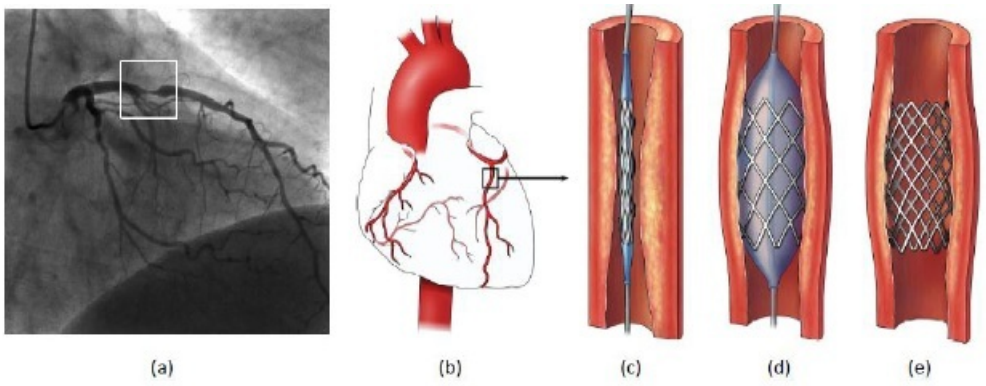


Figure 1.1: (a,b) A narrowed coronary artery of the heart. (c) A stent is inserted through a balloon catheterization. (d) The balloon is inflated, expanding the stent and widening the artery. (e) The stent holds the artery open (Albers, 2010)

.

1.2.2 The Allura X-Ray System

Allura X-Ray systems are interventional systems from Philips Healthcare honed to meet precise cardiovascular, electro-physiology, radiology, neuro-radiology needs[3]. The system has the flexibility to handle a wide range of mainstream diagnostic and interventional procedures with exceptional clarity and deep insight. Figure 1.2 shows an Allura X-Ray system used in a hospital for an interventional treatment. The Allura system has a C-arm that rotates in a curved trajectory around the patient. One side of this arm generates X-Rays using a generator and the other end captures the images using a detector. Physicians perform minimal invasive interventional procedure by viewing the X-Ray images of the patient on the monitor.

Figure 1.2: Allura X-Ray in a hospital

Data flow

The functionalities of the system are explained in brief using a data flow representation [4]. Figure 1.3 shows a conceptual dataflow diagram for Allura X-Ray system.

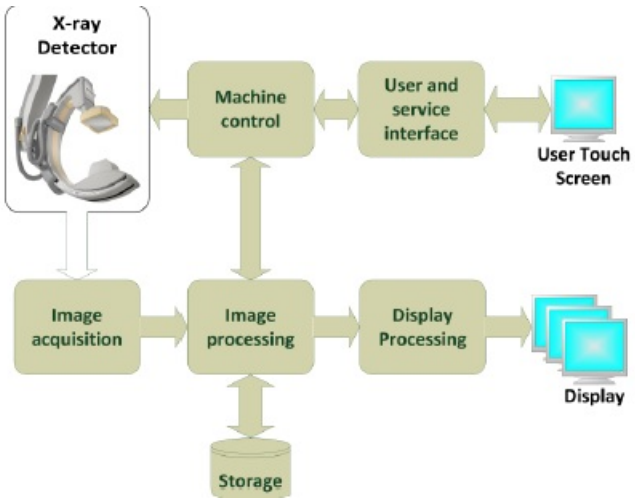


Figure 1.3: Dataflow diagram for Allura X-Ray system (Albers, 2010)

The blocks shown in Figure 1.3 are briefly described below.

- **X-Ray detector:** This is the module that captures the X-Ray images generated once the radiations pass through the patient. The C-arm has both the generator and the detector.
- **Image Acquisition:** The detector sends the data to the image acquisition module

where an X-ray image is constructed. This module does preprocessing on the acquired images before they are sent to the image processing module for further processing.

- **Image Processing:** The images are acquired using 2 modes of X-Ray generation which are, fluoroscopy run in which an extremely low dose of radiation is used and exposure run in which higher dose of radiation is used. The image processing module provides functionalities such as noise reduction, contrast enhancement, image analysis, feature enhancement, motion compensation, and image subtraction of the acquired images. These processed images are also stored on disk for future retrievals and analyses. It is important to retain images with extremely high quality using minimum dose of radiation. Image processing techniques ensure sufficient image quality, clarity and resolution.

- **Display Processing:** Display processing is done to display the images on several monitors adorned with menus, and other manipulation functions. It also compensates for differences in monitor contrast and brightness.

- **Display:** This module is the display module which includes multiple small monitors or a single high resolution huge monitor supported by the Flex-vision PC.

- **Machine Control:** This module enables the physicians to rotate the C-arm and move the table.

- **User and Service interface:** This module enables the physicians to configure the displays and manipulate images shown on the displays. The physicians can switch images from one monitor to another for better viewing. They can, in the case of Flex-vision, choose from a set of defined templates. The user and service interface enables physicians to manipulate images by zooming, panning, and moving images from one viewing port to another. This module also provides the interface for servicing the X-Ray systems. Through the service interfaces, service engineers are able to configure, test, and fix the product.

- **User Touch Screen:** This module is a touch screen that provides access to the User and Service interface.

- **Storage:** The images detected using fluoroscopy and exposure runs are stored onto a disk for future retrievals and can be used for diagnostic purposes. Picture Archiving and Communication Systems(PACS)is used to store images in hospitals.

Clinical setting

Figure 1.4 illustrates the clinical work-flow environment used for an Allura system. In this figure, the control room and exam room are shown. The exam room is where the Allura system is placed. X-Rays are generated, detected and displayed on the monitors for physicians and used in interventions. The control room also accommodates monitors for diagnoses and patient information management for physicians and technicians respectively. The technical room (not shown in figure) is where the tier of PCs are maintained using M-Cabinet along with the cooling system.

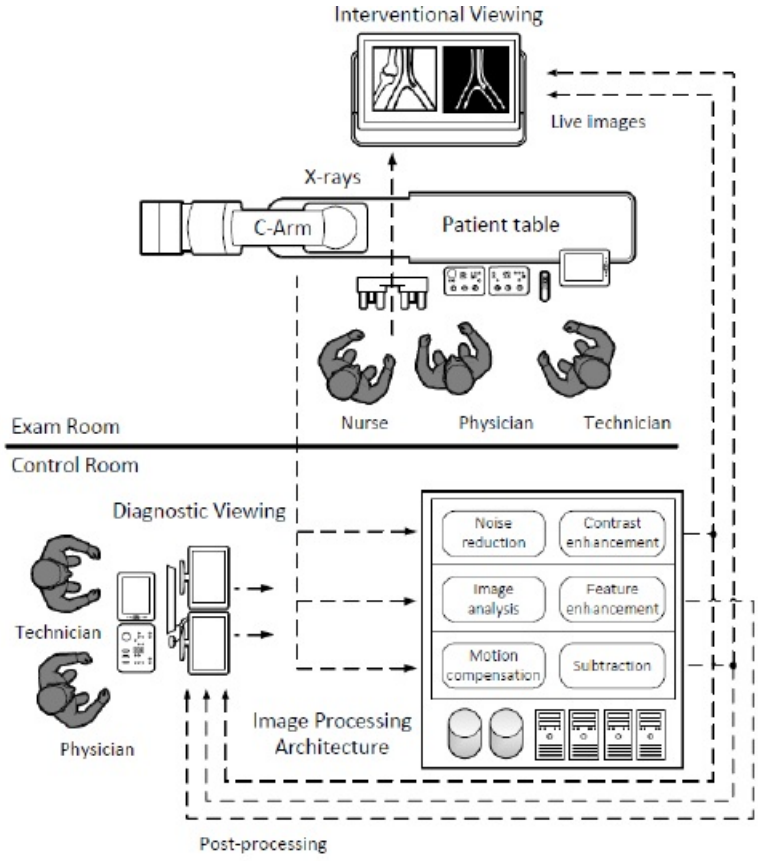
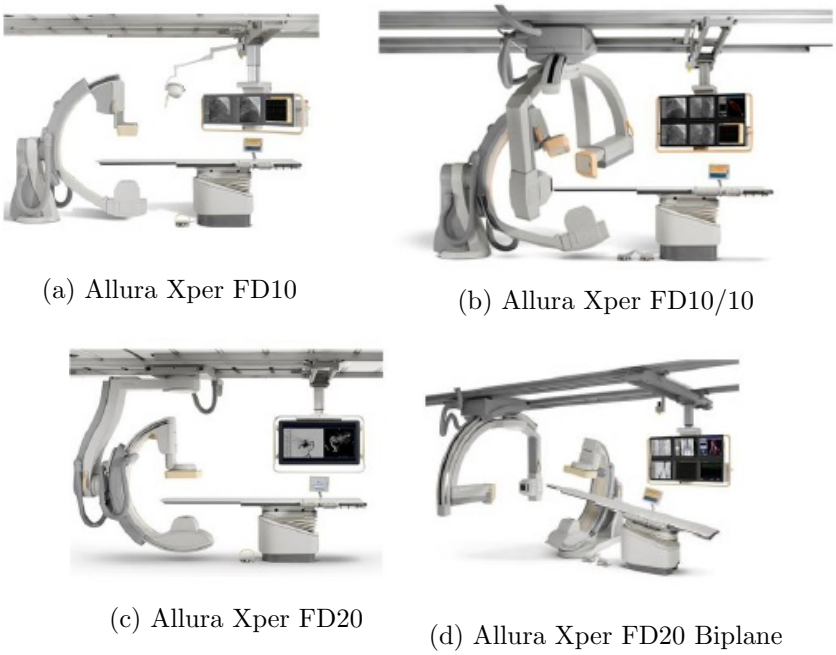


Figure 1.4: Clinical Setting for Allura X-Ray System, in combination with an illustration of the transfer and viewing of applications

Allura Xper Product Series

Interventional Allura X-Ray systems include the monoplane and biplane systems. A biplane system provides twice as much information as the monoplane system by providing both frontal and lateral images unlike the monoplane system that provides frontal images

alone. The series of monoplane and biplane systems come in Xper Product series. They are shown in Figure 1.5



(a) Allura Xper FD10

(b) Allura Xper FD10/10

(c) Allura Xper FD20

(d) Allura Xper FD20 Biplane

Figure 1.5: The Allura Xper Product Series

Figure 1.5a and Figure 1.5c are monoplane systems. Figure 1.5b and Figure 1.5d are biplane systems.

1.3 Scientific Background

As discussed, it is important for physicians to see X-ray images with very low latency on the monitors when performing cardio-vascular minimal invasive interventions. The Image Processing(IP) part of the system should perform extremely well to meet this requirement. Allura X-Ray system combines powerful PCs to meet the requirements of executing a plurality of functions and ensuring guaranteed performance. Performance of the system depends on hardware components, OS and software used to develop the system. Therefore, it is of critical importance to be able to test applications to be run on PCs to predict in advance if the system could be run successfully, before shipping it to the customer. Developing adequate models to be able to test the system will be indeed helpful and thoughtful.

The PCs used in the X-Ray system run an array of tasks and functionalities in parallel.

However, different PCs with exactly the same hardware specification appear to show different levels of resource utilization for the same application. Out of these, some systems succeed in providing the desired levels of performance and the rest, fail. Regular tests happen not to show this. The aim is to find the distinction between these PCs already before the production phase using a test-set derived from the application. This amounts to developing an executable model of the application, that can be handed to the vendor (or manufacturer of the PC).

If we vary the hardware specification, applications also behave differently, in terms of resource usage. The aim further is to predict the execution of an application, depending on the hardware model. This is useful, for example, for the purpose of selecting new hardware and for predicting combining applications onto a single hardware. To that end, descriptive models are developed in which application and hardware properties are parameters. This idea can be used to combine two different PCs into one PC, to determine the optimum configuration for an application and to predict whether the application can run on a PC which is yet to be developed at the vendor's end.

In order to determine the resource usage characteristics of an application on a PC with a particular hardware model, performance properties are measured when the application is being run. The measurements are done using a tool developed at Philips Healthcare called the MeasureLoadCLI tool. The executable models are created using the Resource Usage Modeling Measure Tool (RUMM Tool)[1] designed by Ashenafi Gebreweld at Philips Healthcare.

1.4 Problem Definition

The problem definition of this thesis revolves around two different parts discussed in Section 1.3. They are:

1. *To develop executable models using resource usage characteristics of an application run on a PC to be able to predict the nature of the PC (good/bad[2]) before the production phase*

2. *To develop descriptive models to predict the execution of application(s),depending on the hardware model of the PC*

[1] The RUMM tool is used to generate executable models using micro-benchmark codes derived by creating a profile of application usage and the hardware model chosen.

[2] A good PC is a term used to denote a PC that meets the performance requirements of the Allura X-Ray system and a bad PC is a PC which fails to provide the desired level of performance. PCs that fail to provide the desired level of performance fail due to one or many problems observed during testing, such as, system crash, corrupt images, network card failure, memory problems. Note that, the models intend not to predict easy to find failures,(e.g. defective video card) but instead try to find systems that do not have a consistent failure pattern and seem to work on first sight(hard-to-find failures).

1.5 Contribution

As discussed in Section 1.4, there are two distinct goals in this project. The research questions that are addressed and the solutions to the questions are listed below.

1. What are the hardware metrics that are responsible for the failure of a PC used in the Allura Xper Xray System?

   - Intensive data analysis: Analysis of several hardware metrics, for example, CPU usage, cache misses, cache hits, to observe unusual behaviour in the measurements captured from the PCs.

   - Creation of executable models: Creation of executable models that predict the performance of a PC, on running it on a PC without the need to run the actual application and extensive suite of tests.

2. What are the possible configurations of a hardware model of a PC and the applications run on that PC?

   - Hardware metric analysis: Analysis of hardware metrics is done to observe hardware resource metric usage for a particular application with distinct characteristics. For example, we use an application that stresses memory buses.

   - Creation of descriptive models: Creation of descriptive models that formulate the performance of the applications on a PC. This serves as a basis to predict the performance of the same application(s) on a different PC (different in hardware specification, for example, change in memory configuration).

1.6 Thesis Outline

The remainder of this thesis is organised as follows. In Chapter 2,the problem definition is elaborated to get a deeper insight into the goals of this project. In Chapter 3, the design and development of descriptive models is discussed. It entails the details about the experiments conducted and reflection done on the measurements collected. In Chapter 4, the design and creation of executable models to predict the nature of the PC are discussed in detail. It entails the details and findings during the analysis of the measurements collected. Finally, we conclude the thesis in Chapter 5, and present the future work of this thesis.

Chapter 2

Problem Description

2.1 Introduction

From Chapter 1, it is known that we are interested in determining and predicting the performance of the Allura X-Ray system. We intend to create executable and descriptive models for performance prediction and evaluation. Therefore, it is important to understand the nature of these models before delving into greater depths. This chapter discusses the type of models, the approach used and formalises the problem definition using mathematical notations.

2.2 Solution Approach

Performance model of the architectural instance of a PC can be used to evaluate the performance of the PC in terms of metrics, such as, bandwidth utilisation, throughput of the interconnection technology observed[5]. In this section, we intend to find an approach that evaluates the performance of applications on a PC.

Performance evaluation of the applications running on any PC can be done using the following approaches.

1. Y-Chart Approach: This is a methodology that provides designers with quantitative data obtained by analysing the performance of architectures for a given set of applications[5]. See Figure 2.1. This is described as follows:

    - Architectural Instance: This can also be called Hardware Model/ Platform model of a PC. It specifies the physical, structural and behavioural properties

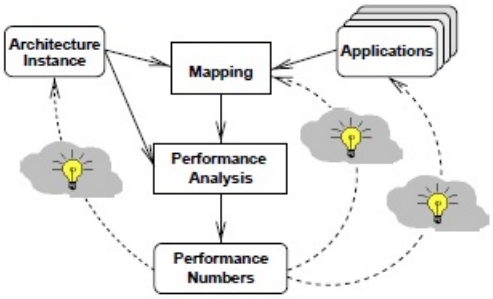


Figure 2.1: Y-Chart Approach

and the performance constraints of the platform used, for example, memory size, bus bandwidth.

- Application Model: This describes the functional behaviour of an application in an architecture-independent manner. It is used to study the target application and obtain estimations of its performance needs, for example, to identify the most expensive computation task. The application parameters are used as input to the application models. The model correctly expresses the functional behaviour, but is independent from the key parameters of architecture constraints, such as, resource utilization or bandwidth limitations.

- Mapping : This is used to map application models on the architectural instance. The mapping helps obtain performance numbers that are useful for performance evaluation.

  The light bulbs shown in Figure 2.1 indicate feedback loops for performance optimization by changing application model, architectural instance and the mapping. However, the object of interest of this thesis is performance evaluation only. Therefore, the parts indicating light bulbs are not important in this context. Though this method provides the performance numbers, the tight coupling of the application models to the application demands new application model creation for every change in the application. This could be time-consuming and expensive. Therefore, this method is not used for performance evaluation.

2. Resource Usage Modeling Approach: This is a methodology that models both the architecture and the application together by analysing the resource usage parameters of the application run on the architectural instances of PCs. The methodology uses a set of tools to measure the resource usage parameters of the application and is used to create models that predict the execution behaviour of the application on another architectural instance (platform). This is exactly what we intend to do. This method eliminates the need to create new application models for modified

applications. Therefore, resource usage modeling approach is chosen for the performance evaluation of the PCs. See Fig 2.2. The star sign in Figure 2.2 indicates that whenever an application is modified, only the resource usage measurements change and the tight coupling between the application behaviour and the models created as seen in the Y-Chart approach is resolved. This makes it easy to maintain and easy to evolve with applications.

Resource usage modeling can be done using one or both of the modeling techniques described.

(a) Execution modeling: An executable model is a piece of executable code that resembles the resource usage of an application (also called micro-benchmarks). The executable models are platform dependent.

(b) Descriptive modeling: A descriptive model is a formula that describes performance metrics of choice as a mathematical formula. Hardware properties take the form of parameters in such a formula(For example, memory size, speeds, number of processors).

In this thesis, we intend to use one or both of the above approaches. They can be used to validate each other.

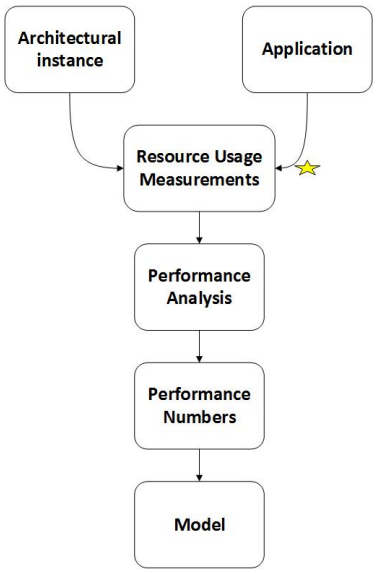


Figure 2.2: Resource Usage Modeling Approach

2.2.1 System Overview

The Allura X-Ray system meets the performance requirements by using a set of computationally efficient PCs. The system has several types of PCs used for user input, image processing, image detection, image display. Not all the PCs are described in this report for brevity. The PCs that are of interest to this thesis are:

1. Host PC: This is the *heart* of the entire system and provides an interface to the user to perform tasks, perform safety checks, conduct tests. This is the only PC that has keyboard, mouse and monitor.

2. Image Processing (IP) PC: This is the PC used for extensive image processing. Images are sampled at a suitable sampling rate and processed to store them for diagnostic purposes. This can be considered as the *brain* of the system.

The IP PCs are more susceptible to performance issues. Tests conducted in the factory reveal that IP PCs fail during performance testing. Also, IP PCs have a very well defined (soft real time), and predictable behaviour, thus must show less variance in the results. Therefore, we study IP PCs.

2.3 Formalizing the Approach

The sequence of steps followed to achieve the goals discussed in Chapter 1 is represented using a formalized approach in this section. This approach is used to introduce the technology and ensure precise semantic meaning of the specifications.

2.3.1 Part I

Part I of the problem definition is: *To develop executable models using resource usage characteristics of an application run on a PC to be able to predict the nature of the PC (good/bad) before the production phase.*

The set of machines is denoted by M. Each machine $m \in M$ is associated with a vector of physical properties describing the details of its construction (e.g. type and make of its graphics cards, NICs). For the purpose of this work, we focus on the amounts and types of resources which we denote as attributes of m. Table 2.1 lists the attributes of interest of m.

The set of programs is denoted by P. Each program $p \in P$ run on m is chosen such that it represents the Allura application.

Table 2.1: Attributes of m

Notation of attribute of m	Description
$m.CPUmodel$	The model of CPU in m
$m.CPUfreq$	CPU frequency in m
$m.Core$	Number of logical cores in m
$m.Socket$	Number of sockets in m
$m.Thread$	Number of threads per core in m
$m.L_0$	Size of L_0 cache in m
$m.L_1i$	Size of L_1 instruction cache in each core in m
$m.L_1d$	Size of L_1 data cache in each core in m
$m.L_2$	Size of L_2 cache in each core in m
$m.L_3$	Size of L_3 cache in m
$m.Mem$	The total DRAM memory in m
$m.MemChannel$	Number of channels from each CPU to DRAM in m
$m.DIMM$	Number of DIMMs per channel
$m.MemFreq$	Memory Frequency in m
$m.TotalMemBw$	Total memory bandwidth per CPU in m
$m.QPIlink$	Number of QPI links between the CPUs in m
$m.TotalQPIBw$	Total QPI bandwidth between the CPUs in m
$m.Hub$	Number of IO hubs in m
$m.PCIeversion$	Type (version) of PCIe used in m
$m.Disk_{SAS}$	Size of the SAS disk in m
$m.Disk_{SATA}$	Size of the SATA disk in m
$m.GraphicCard$	The number of graphic cards in m
$m.GpuMem$	Total amount of memory in the GPU in m
$m.PCIeGraphicBw$	PCIe bandwidth to and from the Graphic card in m
$m.GrabberCard$	Number of grabber cards in m
$m.PCIeGrabberBw$	PCIe bandwidth to and from the Grabber card in m
$m.InfiniBandCard$	Number of Infini-Band cards in m
$m.PCIeICBw$	PCIe bandwidth to and from the Infini-Band card in m
$m.TurboBoostversion$	The Turbo Boost technology version number in m
$m.maxTDP$	Maximum Thermal Design Power (TDP) in m
$m.lithography$	Lithography in m

The notation used is as follows:

- $m1$, $m2$, $\ldots \in M$ denote machines belonging to M
- $p1$, $p2$, $\ldots \in P$ denote programs belonging to P
- $x1$, $x2$, $\ldots \in X$ denote executable models belonging to $X \subseteq P$

- $p1 \triangleright m1$ denotes execution of $p1$ on $m1$
- E is a function: $E : P \times M \rightarrow X$ that is used to create executable models
- $x1 = E(p1 \triangleright m1)$ means that $x1$ is the model derived from $p1 \triangleright m1$
- $k1, \ldots kn \in K$ denote metrics belonging to K. Metrics are functions to real numbers (For example, socket usage, socket L_3 cache miss, disk read, disk writes are metrics in the context of this thesis). $k1(p1 \triangleright m1)$ gives a metric $k1$ on this execution. Similarly, $k2(p1 \triangleright m1)$ gives a metric $k2$ on this execution and so on.
- C denotes a set of classifications of machines into good/bad. $C = \{good, bad\}$
- $k_{ref1}, \ldots k_{refn} \in K_{ref}$ denote threshold (reference) values of metrics for a machine based on its nature (good/bad) belonging to K_{ref}

A sequence of steps are to be followed to derive useful profiles of metrics of machines, compare the profiles with classification made earlier, create executable models and verify their predictive quality. The steps followed are illustrated in Figure 2.3, Figure 2.4, Figure 2.5 using the mathematical notations presented.

From Figure 2.3, it is seen that a program, $p1$ (which represents the application) is selected. It is run on the machines, $m1$ and $m2$ classified based on their nature ((good/bad) known from use and tests). The classification of machines done in this manner, serves as the ground for developing an executable model and verifying its predictive quality. The MeasureLoadCLI tool is also run in parallel with $p1$ to record the Resource Usage Measurement (RUM) of the hardware parameters of interest.

Based on the RUM collected, a profile $k1(p1 \triangleright m1), \ldots kn(p1 \triangleright m1)$ is created for $m1$ and a profile, $k1(p1 \triangleright m2), \ldots kn(p1 \triangleright m2)$ is created for $m2$. A profile is a set of metrics.

The profile of $m1$ is compared with the nature of $m1$ (good/bad) to check whether $p1$ is capable of producing a useful profile. This is done by comparing individual metrics of the profile created for $m1$, $k1(p1 \triangleright m1), \ldots kn(p1 \triangleright m1)$ with the threshold values of the metrics of $m1$ determined based on its nature as shown in Equation 2.1.

$$\begin{aligned} & k_{ref1} - k1(p1 \triangleright m1) \\ & \cap \\ & \vdots \\ & \cap \\ & k_{refn} - kn(p1 \triangleright m1) \end{aligned} \tag{2.1}$$

The qualifying condition for $p1$ to produce useful profile is represented using Equation 2.2. If the weighted sum of the absolute value of the difference between the reference

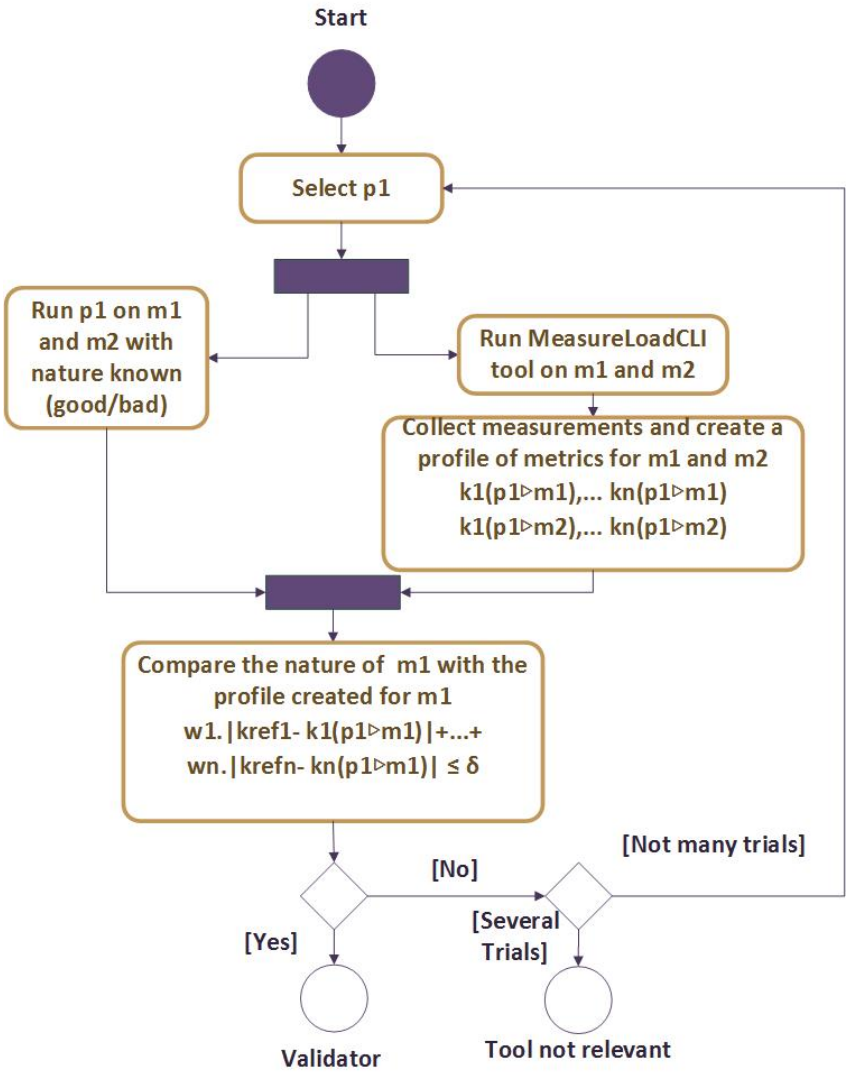


Figure 2.3: Activity Diagram I,Part one out of three: Creating useful profile of metrics for machine chosen

(threshold values) metrics and the profile of the machine is less than the offset selected, δ, then Equation 2.2 holds.

$$w1.|k_{ref1} - k1(p1 \triangleright m1)| + \ldots + wn.|k_{refn} - kn(p1 \triangleright m1)| \leq \delta \tag{2.2}$$

- $w1, \ldots wn$ are the weights assigned to the metrics
- δ is the error offset chosen

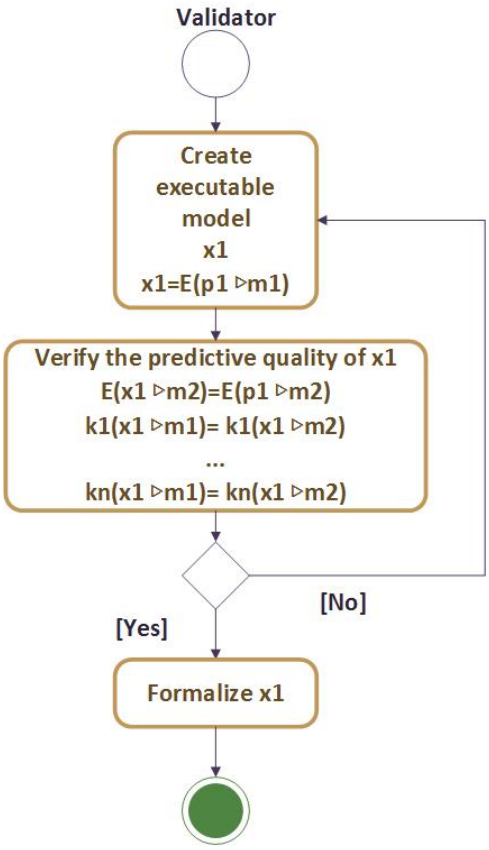


Figure 2.4: Activity Diagram Part I,Part two out of three: Validation

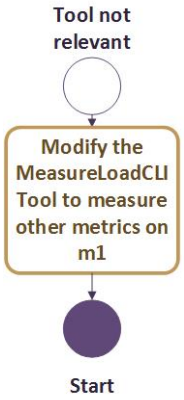


Figure 2.5: Activity Diagram Part I,Part three out of three: Modifying the tool

This step is extended to develop an executable model, $x1$ as seen in Figure 2.4

$$x1 = E(p1 \triangleright m1) \tag{2.3}$$

In order to verify the predictive quality of $x1$, carry out the following steps.

$$E(x1 \triangleright m2) = E(p1 \triangleright m2) \tag{2.4}$$
$$k1(x1 \triangleright m1) = k1(x1 \triangleright m2) \tag{2.5}$$
$$\vdots$$
$$kn(x1 \triangleright m1) = kn(x1 \triangleright m2)$$

Equation 2.4 is used to verify the predictive quality of $x1$ by executing it on $m2$. The equality relation holds true if the execution of $x1$ on $m2$ matches with the execution of $p1$ on $m2$, i.e., if Equation 2.6 holds. In Equation 2.6, individual metrics obtained by the execution of $x1$ on $m2$ is compared with the corresponding metric obtained by the execution of $p1$ on $m2$.

$$k1(x1 \triangleright m2) = k1(p1 \triangleright m2) \tag{2.6}$$
$$\vdots$$
$$kn(x1 \triangleright m2) = kn(p1 \triangleright m2)$$

Equation 2.5 is used to make closer observations and comparison of individual metrics obtained by the execution of $x1$ on $m1$ and $x2$ on $m2$. The equality relation of Equation 2.5 should hold true when $m1$ and $m2$ are of the same nature (e.g., both are good machines) and fail if $m1$ and $m2$ are not of the same nature (e.g., $m1$ is good and $m2$ is bad). If this condition is satisfied, then it means that $x1$ is capable of predicting the nature of the machine and the predictive quality of $x1$ has been verified.

Based on the results of verifications, $x1$ is formalized as an executable model to predict the nature of a PC before production.

If Equation 2.2 fails, then an alternate program, $p \neq p1 \in P$ is used to create a useful profile for $m1$. If Equation 2.2 fails even after repetitive executions of this step, then the MeasureLoadCLI tool is modified to determine measurements of additional/alternate metrics that are to be included in the model. For example, disk access latency, QPI bandwidth are alternate metrics that could be of use, but, not incorporated already in the MeasureLoadCLI tool.

2.3.2 Part II

Part 2 of the problem definition is: *To develop descriptive models to predict the execution of the application(s), depending on the hardware model of the PC.*

The first step in realizing this part of the assignment is to define the hardware model of a machine which is represented as a vector of physical properties. For example, the

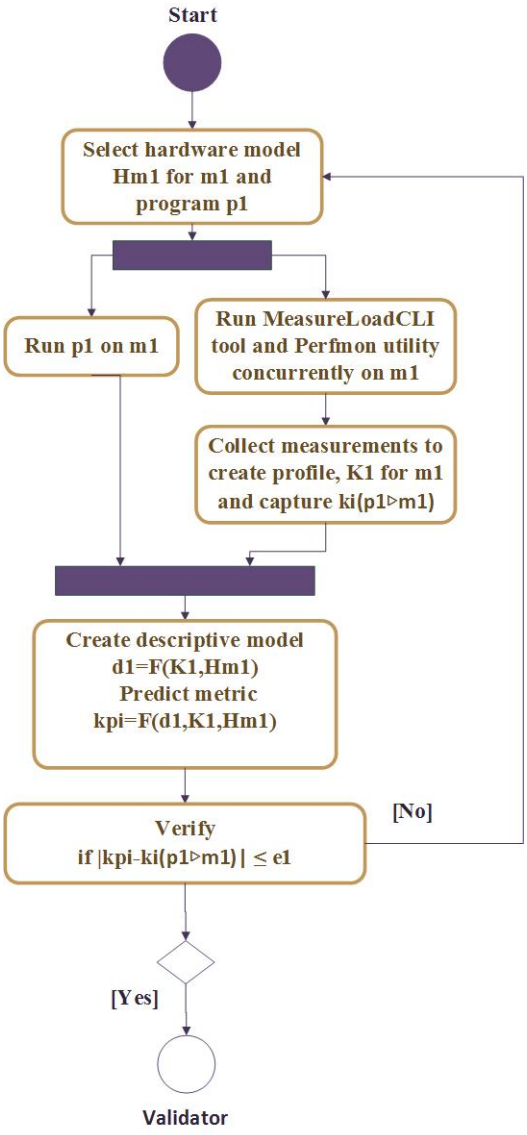


Figure 2.6: Activity Diagram II, One out of two: Representing the sequence of steps to be followed in creating and verifying the descriptive model

hardware model of $m1$ is specified as:

$$\hat{Hm_1} = \begin{pmatrix} m1.Core & m1.Socket & m1.Thread & m1.L_1 & m1.L_2 & m1.L_3 & m1.Hub & \dots \\ 8 & 2 & 1 & 32KiB & 256KiB & 8MiB & 2 & \dots \end{pmatrix}$$

$\hat{Hm_1}$ is a vectorial representation of the physical properties of $m1$. It is also the hardware model of $m1$. Hardware model is similarly defined for $m2$.

The hardware model is used as a basis to create a descriptive model.

- $d1$, $d2$, $\ldots \in D$ denote descriptive models belonging to D
- F is a function: $F : P \times M \rightarrow D$ used to create descriptive models
- $d1 = F(K1, H\hat{m}_1)$ means that $d1$ is the model derived as a function of $K1$ generated on $m1$ using $H\hat{m}_1$. Here, $K1 \subseteq K$ is the profile created on $m1$ such that, $\forall k \in K1$, $k : p1 \times m1 \rightarrow K1$
- Similarly, $d2 = F(K2, H\hat{m}_2)$ means that $d2$ is the model derived as a function of $K2$ generated on $m2$ using $H\hat{m}_2$. Here, $K2 \subseteq K$ is the profile created on $m2$ such that $\forall k \in K2$, $k : p1 \times m2 \rightarrow K2$.

The steps taken to create and verify the predictive quality of the descriptive models is illustrated in Figure 2.6. In order to be able to study the behaviour of the application on $m1$, a program, $p1$, that generates the worst case of load possible on the machine is selected. The MeasureLoadCLI Tool is run in parallel when $p1$ is run on $m1$ to capture RUM of hardware parameters chosen and create profile, $K1$ for $m1$. The MeasureLoadCLI tool also captures a metric, $ki(p1 \triangleright m1) \notin K1$.

Using the profile created for $m1$ and $m2$, develop descriptive models, $d1$ and $d2$.

$$d1 = F(K1, H\hat{m}_1) \tag{2.7}$$

$d1$ created using Equation 2.7 is used for prediction. $d1$ is used to predict $ki(p1 \triangleright m1) \notin K1$ on $m1$.

$$k_{pi} = F(d1, K1, H\hat{m}_1) \tag{2.8}$$

k_{pi} denotes the predicted value of $ki(p1 \triangleright m1) \notin K1$.

The predictive quality of $d1$ is verified by comparing $ki(p1 \triangleright m1) \notin K1$ measured with k_{pi} predicted using Eqn 2.8.

$$|k_{pi} - ki(p1 \triangleright m1)| \leq e1 \tag{2.9}$$

If Equation 2.9 holds, then the metric predicted is comparable with the actual value of the metric measured. This verifies the predictive quality of the model, $d1$. It can be formalized. Taking the error factors into consideration, the metric, k_{pi} is considered acceptable on $m1$ if it is comparable to $ki(p1 \triangleright m1)$ within an error offset of $e1$.

If Equation 2.9 fails to hold, then, $d1$ is re-developed using alternate metrics measured.

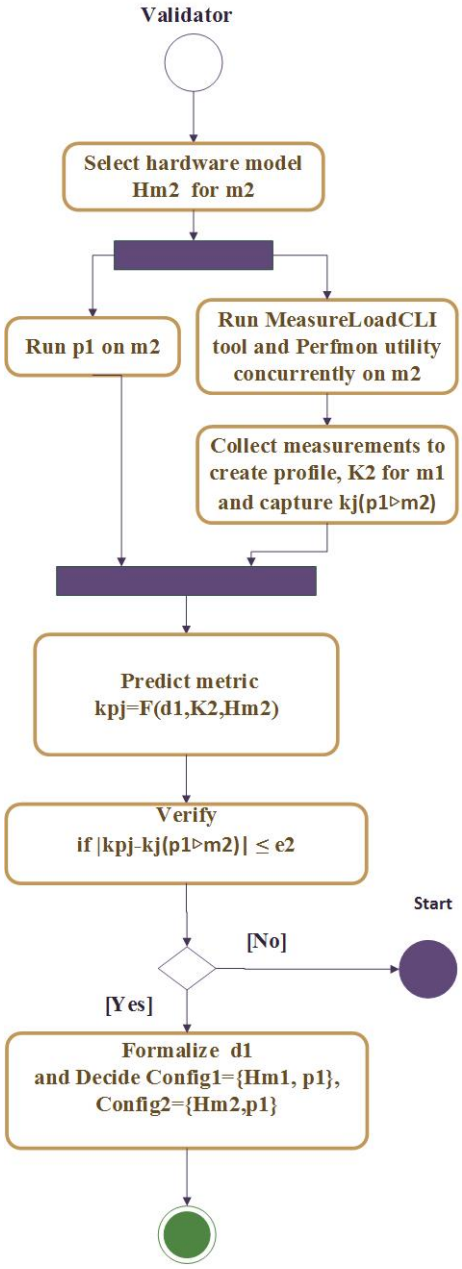


Figure 2.7: Activity Diagram II, Two out of two: Representing the sequence of steps to be followed in validating the descriptive model

The steps taken to validate the descriptive model using another hardware platform is illustrated in Figure 2.7. The predictive quality of the model, $d1$ is validated on another machine, $m2$. $p1$ is run on machine, $m2$. The MeasureLoadCLI and the Perfmon tool are run simultaneously to collect RUM of the hardware parameters which is used to

create a profile of metrics, $K2$ for $m2$. The MeasureLoadCLI tool is also used to capture a metric, $kj(p1 \triangleright m2) \notin K2$.

$$k_{pj} = F(d1, K2, \hat{Hm_2}) \tag{2.10}$$

k_{pj} denotes the predicted value of $kj(p1 \triangleright m2) \notin K2$.

The predictive quality of $d1$ is validated by comparing $kj(p1 \triangleright m2) \notin K2$ measured with k_{pj} predicted as shown in Equation 2.11.

$$|k_{pj} - kj(p1 \triangleright m2)| \leq e2 \tag{2.11}$$

Taking the error factors into consideration, the metric, k_{pj} is considered acceptable on $m2$ if it is comparable to $kj(p1 \triangleright m2)$ within an error offset of $e2$. If Equation 2.11 fails to hold, then, $d1$ is not not validated on $m2$. This means that the predictive quality of $d1$ is limited to machine, $m1$. $d1$ is redeveloped taking alternate metrics into consideration. If Equation 2.11 holds true, then the predictive quality of $d1$ is validated on $m2$. $d1$ can be used to predict performance metrics on different hardware platforms.

If the predictive quality of $d1$ is verified and validated, configurations possible are selected and formalized. For example, plausible configurations are:

$$Configuration1 = \langle \hat{Hm_1}, p1 \rangle$$

$$Configuration2 = \langle \hat{Hm_2}, p1 \rangle$$

This implies that it is possible to execute $p1$ on $m1$ and $p1$ on $m2$ and the properties of $p1$ on $m1$ and $m2$ can be predicted in terms of hardware metrics.

Chapter 3

Descriptive models for resource usage characteristics of applications

3.1 Introduction

Performance prediction and measurement approaches for component-based software systems help software architects to evaluate their systems based on component performance specifications created by component developers[6]. But, prediction methods for performance and reliability of general software systems are still limited and seldomly used in industry[7]. The diverse information needed for the prediction of extra functional properties of a system makes it a challenging task. But, if suitable models can be created to reflect the changing context, such as, different components(components with different specification), different allocation of resources or usage context, they could be used to predict the performance of a system. This can, for example, result in significant cost reduction in a company like Philips Healthcare if the optimum configuration of a system with known hardware platform and usage profile can be discovered using the model created. This chapter summarises the creation of such performance predicting descriptive models using the resource usage profiles of a chosen application run on selected PCs.

The performance of the components in a system is influenced by its usage[7]. The resource demand may vary depending on input parameters (for example, uploading larger files produces a higher demand on hard disk and network). Using this as a keypoint, resource usage metrics of an application (input) are used to create models that serve as a basis to predict the performance of the application on a different hardware platform.

Modeling a system using performance metrics is an abstract concept. It covers umpteen number of metrics for an array of hardware types and specifications. It would be indeed interesting to model the system based on several QoS metrics and exploring which metric can be best used as a predictor. Taking the practical and environmental limitations into consideration, we have decided to model the PC at Philips Healthcare using two cases.

1. Case 1: Compare the performance of the same PC with different memory configurations in terms of latency of an application

2. Case 2: Compare the performance of two PCs with different micro-architectures in terms of latency of an application

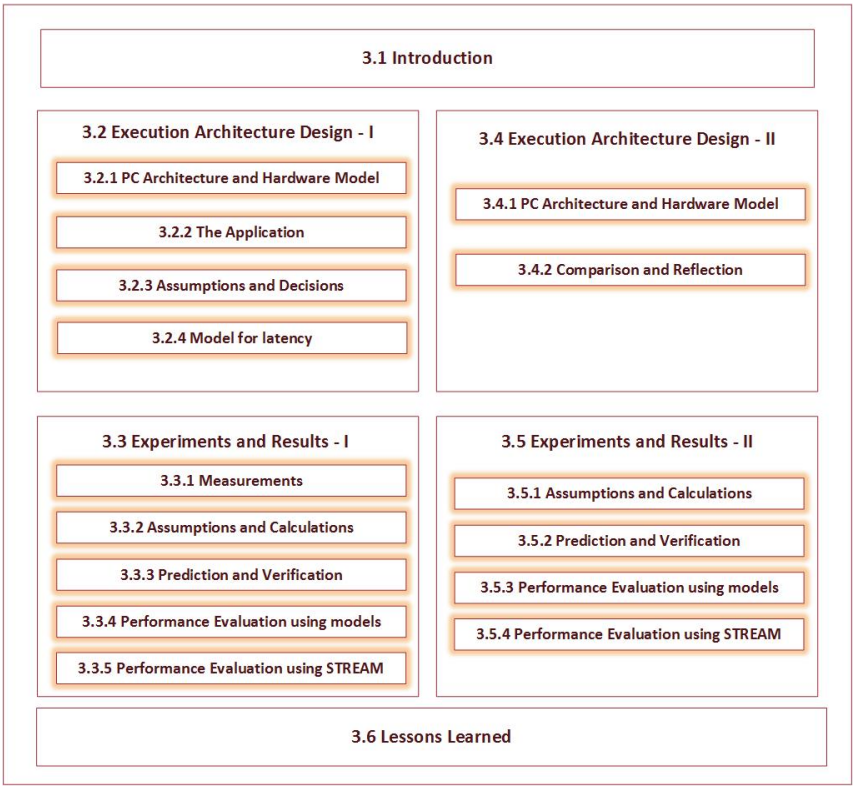


Figure 3.1: Structure of Chapter 3

We will delve into more details of the two cases in different sections. As seen in Figure 3.1, Section 3.2 and Section 3.3 entails the details of Case 1. Section 3.4 and 3.5 entails the details of Case 2. Section 3.6 presents the lessons learned.

3.2 Execution Architecture Design-I

In this section, the application characteristics and requirements to obtain accurate performance estimations of the application on a selected hardware platform are presented for case 1. The execution architecture design encompasses the application properties, hardware model specifications and the mapping of the application on the hardware model.

3.2.1 PC Architecture and Hardware Model

The pictorial representation of the generic hardware model of the PC with Nehalem micro-architecture used for the study in this part of the thesis is shown in Figure 3.2. This hardware model relates to the physical layout of the hardware resources on the motherboard. An instantiated hardware model of a CPU is also depicted in Figure 3.3. The available resources of the instantiated hardware model of the PC chosen, m and its capacity are expressed in Table 3.1.

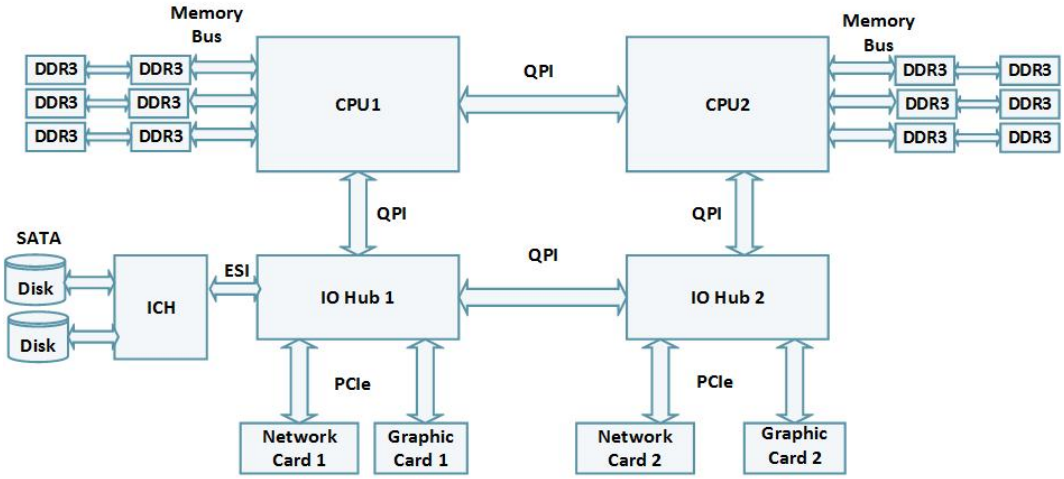


Figure 3.2: Hardware Model of the PC: Case 1

The PC architecture consists of two CPUs running at 2 GHz. Each CPU has four cores. Each core is connected to a private L_1 data and instruction cache of 32 KiB[1] each and a private L_2 cache of 256 KiB. Each CPU is connected to an L_3 cache of 4 MiB which is shared among 4 cores. A pair of memory buses connect each of the 4 cores to an external memory of upto 3 GiB. There are a total of 7 PCIe slots out of which 4 are connected to 2 network cards and 2 Graphic cards. Each Graphic card has an external memory

[1]KiB $\neq$ KB; 1KiB=1024 B whereas 1 KB= 1000 B. Memory capacity is expressed in KiB and Memory Bus Traffic in KB. Similarly, 1MiB=1024 $*$ 1024 B and 1 GiB=1024 $*$ 1024 $*$ 1024 B

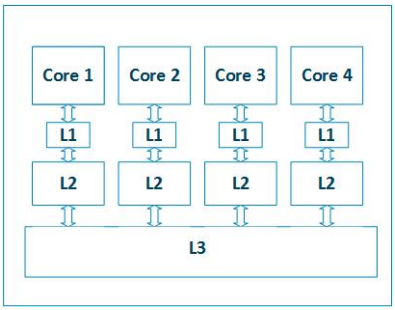


Figure 3.3: CPU model:Case 1

Table 3.1: Attributes of the hardware model of the PC

Attribute of machine m	Type/Capacity
$m.CPUmodel$	Intel Xeon CPU E5504
$m.Core$	8
$m.Socket$	2
$m.Thread$	1
$m.L_1i$(KiB)	32
$m.L_1d$(KiB)	32
$m.L_2$(KiB)	256
$m.L_3$(MiB)	8
$m.MemChannel$	3
$m.DIMM$	2
$m.MemFreq$(MHz)	800
$m.Disk_{SATA}$(GiB)	150
$m.Hub$	2
$m.PCIeversion$	2.0
$m.GraphicCard$	2
$m.GpuMem$(MiB)	512
$m.PCIeGraphicBw$ (Gbps)	80
$m.NetworkCard$	2
$m.PCIeNCBw$(Gbps)	2.5 and 10
$m.NetworkCard$	2
$m.InfiniBandCard$	1
$m.PCIeICBw$(Gbps)	20
$m.lithography$(nm)	45

of 512 MiB. IOHub1 is connected to two SATA disks with 16 MiB cache and 150 GiB capacity each.

3.2.2 The Application

As presented, the aim of part II of the thesis is in principle, to create descriptive models that admit the behaviour of the original Allura application. However, this a huge step due to the complexities and plurality of functions implemented in Allura application. Therefore, we go for synthetic benchmarks that provide measurements with notable resource usage on specific hardware components. Selecting a suitable program, $p1$ is the first step as seen in the activity diagram in Chapter2.

Taking the performance metrics required for Case 1 into consideration, programs that stress the memory buses and generate significant data traffic to the memory are required. The applications used to generate load on the PC were:

- Prime95 Test: Prime95[8] is an application software that has a feature called Torture Test that allows maximum stress testing on the CPU and RAM. There are several options allowing the stress test to focus on the memory, processor, or a balance of both. This test was initially used and the memory bus traffic was observed. As the application fails to generate sufficient traffic on the memory bus, it is not used as a benchmark for creating the model.

- STREAM Benchmark: The STREAM benchmark[9] is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. This application generates sufficient traffic on the memory bus by working on datasets much larger than the available cache on any system. Therefore, we chose STREAM benchmark.

STREAM

- STREAM is intended to measure the bandwidth from main memory. It measures the performance of four long vector operations. These operations are listed in Table 3.2.

- The operations in Table 3.2 are representative of the building blocks of long vector operations. The array sizes are defined so that each array is larger than the cache of the machine to be tested, and the code is structured so that data re-use is not possible.

- Each of the four tests adds independent information to the results:

  - *Copy* measures transfer rates in the absence of arithmetic.

  - *Scale* adds a simple arithmetic operation.

  - *Sum* adds a third operand to allow multiple load/store ports on vector machines to be tested.

  - *Triad* allows chained/overlapped/fused multiply/add operations.

- The general rule[10] for STREAM is that each array must be at least 4 times the size of the sum of all the last-level caches used in the run, or 1 Million elements, whichever is larger. For example, on the dual socket PC with 8 MiB L_3 cache we used, each array needs to be of $max(4 * 8$ MiB/8B, $10^6)$ elements which is equal to 4 million elements.

Table 3.2: STREAM vector kernel operations

name	**Vector Kernel**
COPY	a(i)=b(i)
SCALE	a(i)=q * b(i)
SUM	a(i)= b(i)+c(i)
TRIAD	a(i)= b(i)+q * c(i)

STREAM Application steps
The application traverses the specified usage scenario (array size and offset) from the start action to the stop action for specified number of times. The steps for one vector kernel operation (COPY) are summarized in Figure 3.4.

In Figure 3.4, the steps traversed from the start action to the stop action are depicted. Sum, Add and Triad vector kernel operations are also performed in the same fashion. The peak memory bandwidth for every vector kernel operation is calculated. Note that STREAM uses 8 threads (same as the number of cores on the PC chosen) in parallel for memory allocation and vector kernel operations. STREAM is a command-line application software and can be tuned to accept user input for N, NTIMES and OFFSET where,

- N indicates array size (number of elements in the array)

- NTIMES indicates the number of trials to be performed

- OFFSET is the parameter used to achieve peak bandwidth value by adjusting N(not used in the context of this project)

As we are interested in performance modeling of the PC with different memory configurations, STREAM is used to measure memory bus traffic by choosing the array size carefully.

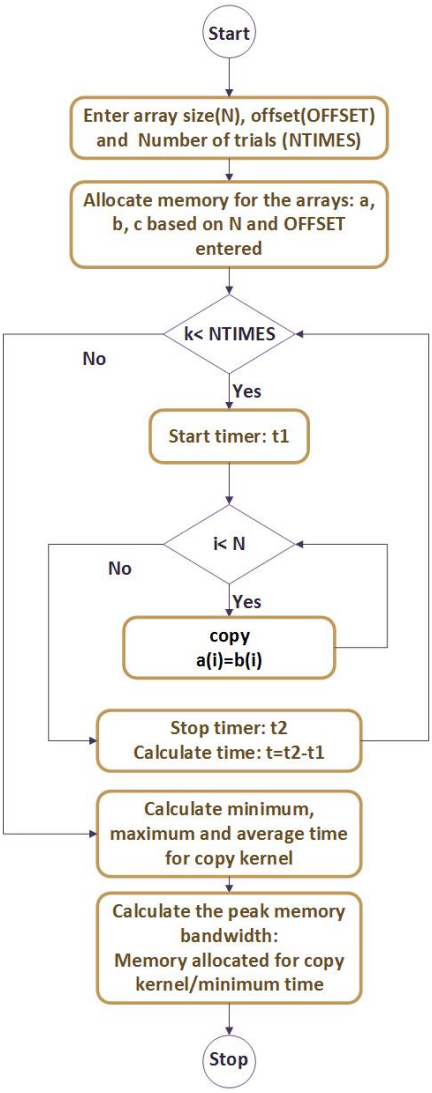


Figure 3.4: STREAM Flowchart

3.2.3 Assumptions and Decisions

There are several assumptions used and decisions made in the course of model creation and the use of accompanying tools. The most important ones are listed below.

- *Number of Sockets:* The analysis makes use of a dual socket machine (IP PC). Metrics such as QPI traffic which are required for creating the model (will be seen later) can be captured only when both the sockets are identified by the OS. For this, it is required to turn the Non Uniform Memory Access(NUMA) on in the BIOS setting of the PC.

- *Measurement Tool:*
  - The MeasureLoadCLI tool is used to capture resource usage metrics such as core usage, core L_3 cache miss, core IPC, network bytes sent and received. The RUMM Tool can also be used to measure metrics such as socket usage, QPI traffic, memory bus traffic. But, it only gives coarse level measurements of metrics without including low level metrics such as the core usage, cache hits and misses. Moreover, the timer used in RUMM Tool supports a granularity of $1s$. MeasureLoadCLI can support a granularity of $1ms$. Taking these factors into consideration, the MeasureLoadCLI tool is chosen for RUM.
  - Perfmon: Perfmon[11]is used to collect windows performance metrics such as hard page faults, page reads, disk queue length that are relevant to the model creation.

    Refer to the activity diagram in Chapter 2 to see the stages in which the tools are used.

3.2.4 Model for latency

This section presents the details of the creation of descriptive models which can be used to predict/estimate the latency of an application . The latency of the selected application in this context is the total running time of the application. It can be defined as the total time spent on all the steps traversed between the start and the stop action. It is estimated in terms of the cache, memory and disk access time (if any). The part of the hardware model studied is highlighted in Figure 3.5 using a red dotted box. The parts highlighted in green are those metrics which are measurable[2]. Note that there are many other metrics measured using Perfmon and MeasureLoadCLI tool which are not significant in the context of this part of the thesis.

The hardware and processor models are adopted from Figure 3.2 and Figure 3.3.The hardware specification of the PC is listed in Table 3.1

The hardware model shown in Figure 3.5 can be studied from a high level using a redefined simplified representation. This includes the *processors*, L_1, L_2, L_3, $Memory$ and $Disk$. See Figure 3.6.

The metrics that are captured using MeasureLoadCLI are obtained using the Intel PCM 2.3. If $M(i,j)$ represents the number of j level cache misses in the core i, $H(i,j)$ represents the number of j level cache hits in the core i, $M(p,j)$ indicates the number of j level cache misses in a processor(CPU) p and $H(p,j)$ indicates the number of j level

[2]Only the incoming QPI metrics are measurable on a Nehalem PC

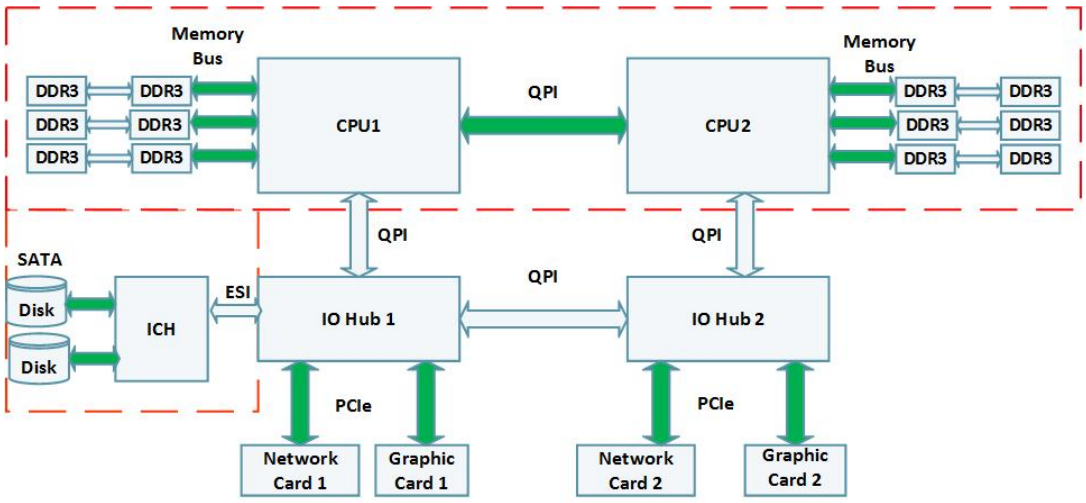


Figure 3.5: Hardware Model used for Study

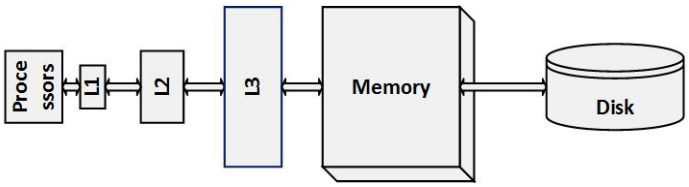


Figure 3.6: Hardware Model used for Study

cache hits in a processor p, then intuitively, the interpretation of some terms of interest are presented in Table 3.3.

The interpretation of the terms is based on the fact, that intuitively we assume the caches to be *exclusive*[3]. However, the actual definitions of the counters according to the Intel counters used in the PCM differ from those presented in Table 3.3. This difference can be explained because of the cache organisation on a Nehalem PC.

L_1 Cache : At Level 1, separate instruction and data caches are part of the Nehalem core. The instruction and the data cache are each 32 KiB in size. The instruction and the data caches have 4-way and 8-way set associative organization, respectively.

L_2 Cache : Each core also contains a private, 256 KiB, 8-way set associative, unified level 2 cache (for both instructions and data). The write policy is write-back and the cache is *non-inclusive*.

L_3 Cache: The Level 3 cache is a unified, 16-way set associative, 8 MiB *inclusive* cache shared by all four cores on the chip.

[3]In exclusive cache, data is guaranteed to be in utmost one of the cache levels

Table 3.3: Interpreting Cache terminology: $p = 1, 2$ and $i = 1, \ldots 8$ on the PC selected

Term	Formula	Interpretation
$L_3misses$	$\sum_{p=1}^{2} M(p,3)$	Number of L_3 cache requests that results in a miss in the L_3 cache only
$L_2misses$	$\sum_{i=1}^{8} M(i,2)$	Number of L_2 cache requests that results in a miss in the L_2 cache only
L_1miss	$\sum_{i=1}^{8} M(i,1)$	Number of L_1cache requests that results in a miss in the L_1 cache only
L_3hits	$\sum_{p=1}^{2} H(p,3)$	Number of requests to L_3 cache that result in a hit in the L_3 cache only
L_2hits	$\sum_{i=1}^{8} H(i,2)$	Number of requests to L_2 cache that result in a hit in the L_2 cache only
L_1hits	$\sum_{i=1}^{8} H(i,1)$	number of requests to L_1 cache that result in a hit in the L_1 cache only

In order to understand the definitions used by Intel PCM, we need to distinguish between the nature of the cache used. Contrary to the assumption, the L_3 cache is *inclusive*(unlike L_1 and L_2), meaning that a cache line that exists in either L_1 data or instruction, or the L_2 unified caches, also exists in L_3 cache. The L_3 is designed to use the inclusive nature to minimize[12] *snoop*[6] traffic between processor cores and processor sockets. A cache block in use by a core in a socket is cached by its L_3 cache which can respond to snoop requests by other chips, without disturbing (snooping into) L_2 or L_1 caches on the same chip.

From Figure 3.7, it can be seen that if the cache is *exclusive*, then a miss in the L_3 cache needs a check in the other core's L_2 cache as L_3 does not duplicate the contents of L_2 caches of individual cores[12, 13]. On the contrary, L_3 cache miss on an inclusive cache guarantees that the data is not on the die because the contents of L_2 are duplicated in L_3. Note that *non-inclusive*[7] and *exclusive* do not imply the same meaning.

See Table 3.4 for the definitions from PCM.

The Intel PCM does not provide counters that measure L_1 cache metrics. As L_1 cache metrics are significant in the context of creating models, they are used in the model. $L_1HitRatio$ is assumed and L_1Hit, L_1Miss are estimated. The definitions of L_1 cache metrics used in the model are listed in Table 3.5.

[4]For any core, sibling core is a core on the same die or socket

[5]i=1, . . . , 4 indicates number of cores in a processor; p=1, 2 indicates the number of physical processors

[6]The processors have the ability to eavesdrop the address bus for other processor's accesses to system memory and to their internal caches called snooping. They use this snooping ability to keep their internal caches consistent both with system memory and with the caches in other interconnected processors.

[7]In a non-inclusive cache, there is no enforcement of cache inclusion or exclusion. A cache line in an inner cache may or may not be in the outer cache

Table 3.4: Hardware Counters from Intel PCM

Counter	**Formula**	**Definition**
L_2HitM	for p=1, 2 { for $i = 1 \ldots 4$ { $\sum_{k \neq i}$ H(k,2) } }	Total number of L_2 cache requests to every core that result in a hit in a sibling [4] core's L_2 cache in every processor. [5](L_3 being inclusive, includes this hit)
$L_3UnsharedHit$	-	Number of L_3 cache requests that result in a hit in the L_3 cache with no snooping required
L_2Hit	L_2hits	Number of L_2 cache requests that result in a hit in the L_2 cache only
L_3Hit	$L_2HitM + L_3UnsharedHit$	Sum of the total number of L_2 cache misses that result in a hit in a sibling core's L_2 cache (L_3 being inclusive, has a copy) and the number of hits in L_3 cache that requires no snooping
L_3Miss	$L_3misses$	Number of L_3 cache requests that do not result in a hit in the L_3 cache
L_2Miss	$L_2HitM + L_3UnsharedHit + L_3Miss$	Total Number of L_2 cache requests to every core that do not result in a hit in the same core's L_2 cache. $L_2Miss \neq L_2misses$
$L_2HitRatio$	$\frac{L_2Hit}{L_2Hit + L_2Miss}$	Ratio of the total number of requests to the L_2 cache that result in a hit in the L_2 cache to the total number of requests to the L_2 cache
$L_3HitRatio$	$\frac{L_3Hit}{L_3Hit + L3Miss}$	Ratio of the total number of the requests that result in a hit in the L_3 to the total number of requests to the L_3 cache

The Venn diagram representation of the metrics relevant to the model (and used in PCM) is shown in Figure 3.8. Note that the diameter of the circles are just an indication of the relative numbers of the metrics obtained from STREAM. (It could be different for different applications.)

In order to estimate the latency of the application, it is important to determine the access times to the cache and memory organisation. The access times to caches are not

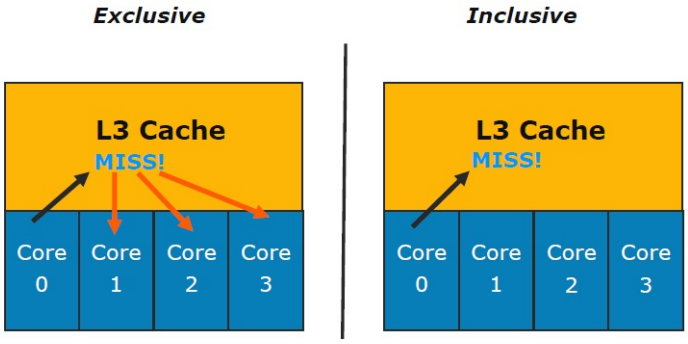


Figure 3.7: Exclusive versus Inclusive cache (Semin, 2009)

Table 3.5: L_1 cache metrics

Metric	**Formula**	**Interpretation**
L_1Hit	L_1hits	Number of L_1cache requests that result in a hit in the L_1 cache only
L_1Miss	$L_2Hit + L_2Miss$	Total Number of L_1 cache requests that do not result in a hit in the L_1 cache
$L_1HitRatio$	$\frac{L_1Hit}{L_1Hit + L_1Miss}$	Ratio of the total number of L_1 cache hits to the total number of requests to the L_1 cache

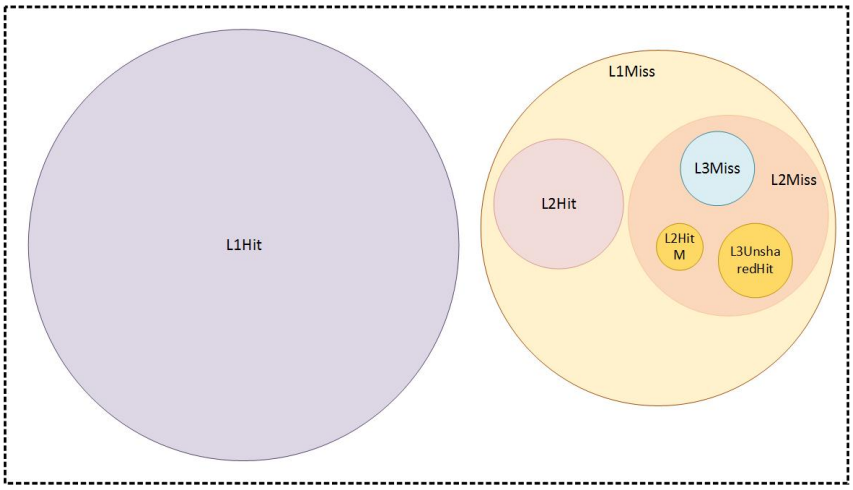


Figure 3.8: Metrics used in the model represented using Venn Diagram

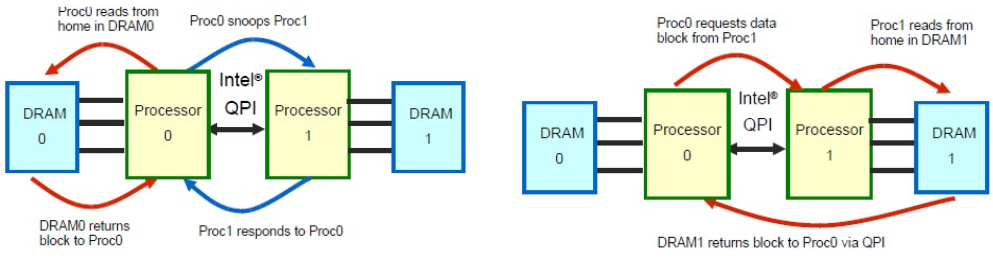
constant values [14, 12, 15]. They depend on various factors such as, core, un-core frequencies, type of access (remote/local, with snooping/without snooping). The following observations and study[14] form the basis for creating this model. See Figure 3.9

Local Memory Access: The steps to access the local memory block are as follows:

1. Proc0 requests a cache line which is not in its L_1, L_2, L_3 cache.
   - Proc0 requests data from its DRAM
   - Proc0 snoops Proc1 to check if data is present there.
2. Response
   - local DRAM returns data
   - Proc1 returns snoop response
   - Proc0 installs block in its L_3, L_2, L_1 cache and retrieves target memory word.

Remote Memory Access: The steps to access the remote memory block are the following:

1. Proc0 requests a cache line which is in not in Proc0's L_1, L_2, L_3 cache
2. Request sent over QPI to Proc1
3. Proc1's probes for cache line
   - Proc1 makes requests to its own DRAM
   - Proc1 snoops internal caches
4. Response
   - Data block returns to Proc0 via the QPI
   - Proc0 installs cache block in L_3, L_2, L_1



(a) Local Memory Access Event Sequence (b) Remote Memory Access Event Sequence

Figure 3.9: Nehalem Memory Access Sequence (Thomakadis, 2011)

From the steps mentioned, it can be inferred that the access latency is a function of QPI

latency and the number of local and remote accesses to memory and caches. The cache coherency protocol messages (snooping) among the multiple sockets are exchanged over the QPI. We have taken these factors into consideration and modelled the cache and memory access times as a function of QPI traffic, local and remote memory access.

The access times to L_2, L_3 and memory are calculated and denoted as $t_{hit2}, t_{hit3}, t_{mem}$. Looking back at Figure 2.6, using STREAM as $p1$, descriptive models are created as a function of the profile of metrics created on the machine and the hardware model of the machine. The set of models $d1, \ldots d4 \in D$ are presented.

$$d1 = F(\{QPI_{sock}, M_r, M_w\}, \hat{Hm}) :$$
$$RemoteAccessRatio(RAR) = \frac{QPI_{sock}}{M_r + M_w} \tag{3.1}$$
$$d2 = F(\{RAR, t_2remote, t_2local\}, \hat{Hm}) :$$
$$t_{hit2} = RAR * t_2remote + (1 - RAR) * t_2local \tag{3.2}$$
$$d3 = F(\{RAR, t_3remote, t_3local\}, \hat{Hm}) :$$
$$t_{hit3} = RAR * t_3remote + (1 - RAR) * t_3local \tag{3.3}$$
$$d4 = F(\{RAR, t_{mem}remote, t_{mem}local\}, \hat{Hm}) :$$
$$t_{mem} = RAR * t_{mem}remote + (1 - RAR) * t_{mem}local \tag{3.4}$$

where ,

- $t_2remote$, $t_3remote$, $t_{mem}remote$ are the access times to remote L_2, L_3, DRAM respectively
- t_2local , t_3local, $t_{mem}local$ are the access times to local L_2, L_3, DRAM respectively.
- M_r indicates the total amount of traffic read from the local and the remote DRAM by the system.
- M_w indicates the total amount of traffic written into the local and the remote DRAM by the system.
- QPI_{sock} indicates the total traffic between the sockets (CPUs) via the QPI link.

The RAR is estimated using the principle that all remote accesses happen via the QPI.

A cache miss in L_3 in one socket finds the data in another socket's L_3 cache (inclusive) or DRAM (local or remote). We have also considered the worst case scenario in which the L_2 cache lines are modified and also requires snooping and thus included remote access latency for L_2 cache. The access time to L_1 is assumed to be constant[15] as L_3 being inclusive accounts for the snooping requests of L_1 cache. The access time to disk (in ms) is extremely high when compared to the QPI latencies (in ns). So, we ignore the QPI latencies when determining the disk latency.

The access times to L_1 and disk are denoted by t_{hit1} and t_{disk} respectively.
Taking the metrics presented in Table 3.5 and Table 3.4 and the models $d1, \ldots d4$ into consideration, the model used to calculate the total latency of the application on a PC with a hardware model , $\hat{Hm}$ is d10 $\in D$ and represented using Equation 3.5.

$d10 = F(d1, \ldots d9, \hat{Hm})$:

$$\boxed{T_{tot} = t_{hit1} * L_1Hit + t_{hit2} * L_2Hit + t_{hit3} * L_3Hit + t_{mem} * MemHit + t_{disk} * DiskHit} \tag{3.5}$$

where, $d5, \ldots d9 \in D$ are presented using the following Equations.

$$d5 = F(PageReads, \hat{Hm}) :$$
$$DiskHit = PageReads \tag{3.6}$$
$$d6 = F(L_3Miss, \hat{Hm}) :$$
$$MemHit = L_3Miss \tag{3.7}$$
$$d7 = F(\{L_3HitRatio, L_3Miss\}, \hat{Hm}) :$$
$$L_3Hit = \frac{L_3HitRatio * L_3Miss}{1 - L_3HitRatio} \tag{3.8}$$
$$d9 = F(\{L_2HitRatio, L_2Miss\}, \hat{Hm}) :$$
$$L_2Hit = \frac{L_2HitRatio * L_2Miss}{1 - L_2HitRatio} \tag{3.9}$$
$$d8 = F(\{L_1HitRatio, L_1Miss\}, \hat{Hm}) :$$
$$L_1Hit = \frac{L_1HitRatio * L_1Miss}{1 - L_1HitRatio} \tag{3.10}$$

DiskHit: This represents the number of disk accesses. This can be estimated using the number of Page Reads/s[16]. Page Reads/s is a memory counter measured using Perfmon. This counter indicates the number of read operations that were required to be done by the disk to retrieve faulted pages (One of the possible scenarios where this is most likely to happen is when the system runs out of memory and accesses the disk).

MemHit: This represents the number of accesses that find data in memory. It is assumed that all L_3Miss result in memory hits.

Looking back at Figure 2.6, the next step is to collect measurements of hardware metrics on selected hardware platforms by conducting experiments. These metrics are used to predict relevant (e.g, runtime) parameters based on just the application properties.

3.3 Experiments and Results-I

Ralph Waldo Emerson quoted,

> *" All life is an experiment. The more experiments you make, the better "*

The experiments are conducted to collect measurements of hardware metrics to create the models presented in Section 3.2.4 and verify its correctness. In this Section, the experiments conducted for case 1 are described which is: *Series of experiments for different memory configurations, which are,* $1GiB, 2GiB, 4GiB, 6GiB$.

The generic sequence of steps followed from creating the models to verifying them are illustrated in Figure 3.10.

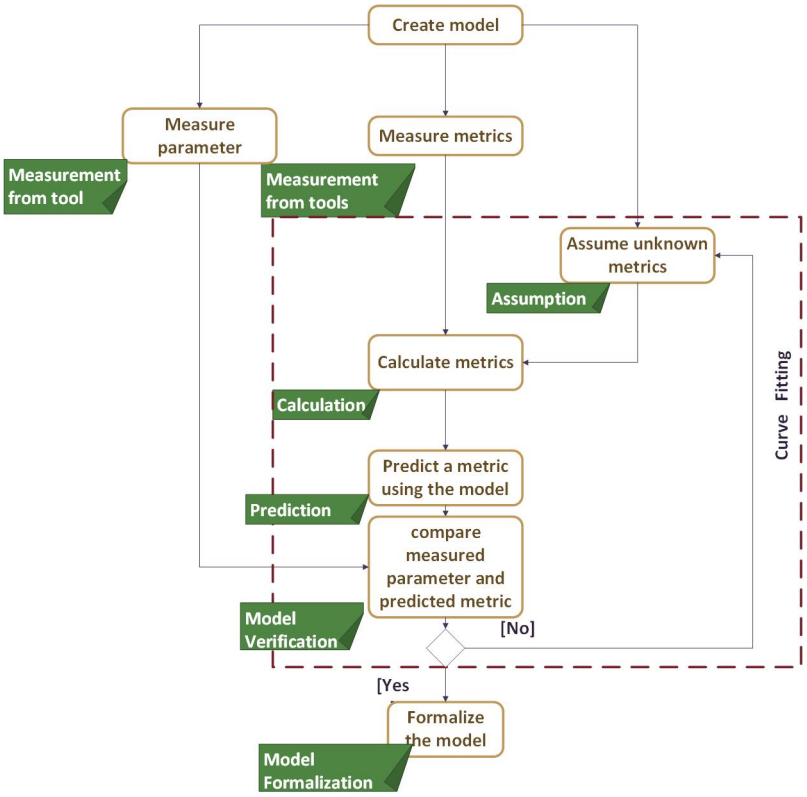


Figure 3.10: Creating models to verifying models: Sequence of steps. The notes (in green) is the name of the action completed in each step.

An instantiation of the sequence of steps is done for model, $d10$.

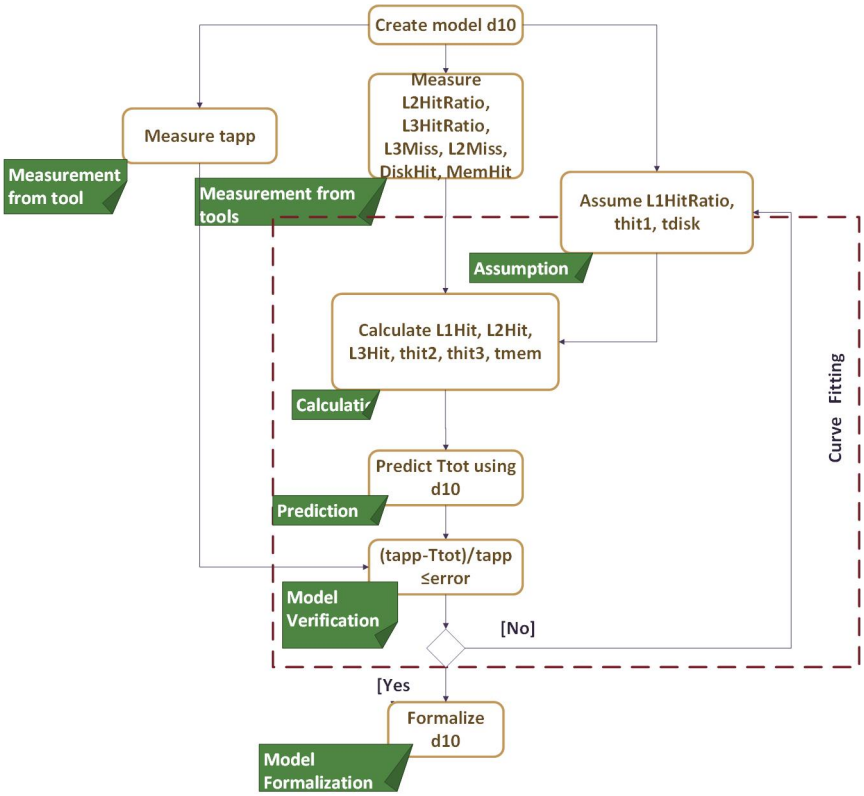


Figure 3.11: Creating model to verifying model, $d10$: Steps

The rest of this section is organised to present individual actions performed in Figure 3.10.

3.3.1 Measurements

The models $d1, \ldots d10$ are created using the measurements from experiments and analysis done in incremental steps. Therefore, the action, model creation shown in Figure 3.10 is not a single step action, but includes a number of steps within itself. The models created are presented in Section 3.2.4. s

The next step in Figure 3.10 is collecting measurements of hardware metrics using the MeasureLoadCLI and Perfmon Tools. The application chosen is STREAM for reasons mentioned in Section 3.2.2.

The experiments are conducted on an IP PC with hardware specifications presented in Table 3.1. STREAM application is run for various array sizes ranging from 1 million elements to 80 million elements. The Measurement Tools, MeasureLoadCLI and Perf-

mon are run concurrently when STREAM is launched. The parameters used in the experiments are listed in Table 3.6. The OS used on the PC is Windows XP.

Table 3.6: Application and Measurement Tool specifications: These are the user input values chosen during the experiments to obtain sufficient samples for creating the model.

Parameter	Value
NTIMES	100
OFFSET	0
Time interval between samples during measurement (s)	1

The list of metrics measured for the series of experiments conducted are tabulated in Table 3.7. The measured values of metrics are tabulated in Tables 3.21, 3.22, 3.23, 3.24.

Table 3.7: Measured Metrics from the tools

Parameter
$L_2HitRatio$
$L_3HitRatio$
L_3Miss
L_2Miss
$DiskHit$
$MemHit$
M_r
M_w
QPI_{sock}

3.3.2 Assumptions and Calculations

This section entails the description of metrics assumed and metrics calculated from assumed metrics and measured metrics. It also describes the rationale behind the choice of important metrics assumed.

The access times to the local and remote caches and memory for a PC with specifications mentioned in Table 3.1 are shown in Table 3.8. The values of the access latencies chosen are based on Intel and Nehalem literature sources[14, 12, 15]. Although, these values are assumed, they are constant for all array sizes chosen.

Access time to remote memory by a processor is approximately 1.6 to 1.7 times the access time to local memory[14]. The latency to remote caches is due to the QPI latency and this is approximately equal to 40 ns[14, 12]. This is used to calculate the access times to remote caches, L_2, L_3 and memory. ($\therefore$, $t_3remote$=t_3local + QPI latency=20+40=60 ns;

Table 3.8: Access Times to Cache and Memory

Cache Level /Memory	**Local**	**Remote**
$L_1(ns)$	t_{hit1}	t_{hit1}
	2	2
L_2(ns)	t_2local	$t_2remote$
	6	46
L_3(ns)	t_3local	$t_3remote$
	20	60
DRAM(ns)	$t_{mem}local$	$t_{mem}remote$
	65	120
	t_{disk}	t_{disk}
Disk(ms)	2	2

Similarly for L_2). The access times to local caches and memory chosen and the access times to remote caches and memory calculated are used to calculate the total access times to L_2, L_3 caches and memory, t_{hit2}, t_{hit3}, t_{mem} for all array sizes chosen. The values of the access times are tabulated for all array sizes in Tables 3.21, 3.22, 3.23, 3.24.

The value of $L_1HitRatio$ is not measured by the tools used. However, the value chosen is not a constant value and ranges from 0.95 to 0.99 depending on the array size. The value chosen is greater than 0.95 because, generally on any PC, the $L_1HitRatio$ is close to 1. A very low $L_1HitRatio$ indicates that the application does not use the cache effectively. The value chosen for different array sizes gives the best regression fit for every experiment. Looking back at Figure 3.11, the value of $L1HitRatio$ chosen becomes a part of curve fitting for models created.

The choice of $L_1HitRatio$ for various array sizes is shown in Figure 3.12. From Figure 3.12, it is seen that value chosen shows variations for larger array sizes. Larger the array size, more the memory required. The memory required is calculated as:
(Maximum number of operands required for the vector kernel operation $*$ Array Size (N) $*$ Number of Bytes per array element)
For example, for an ADD operation with array size of one million elements, the memory required is $= 3 * 10^6 * 8$=22.9 MiB.

The access time to disk ranges from (1 ms-100 ms). We chose 2 ms because this value gives the best regression fit in the model for all possible memory configurations chosen.

RAR is calculated as using the formula represented by model, $d1$.

The metrics assumed and calculated and used in models, $d1, \ldots d10$ are listed in Table 3.9.

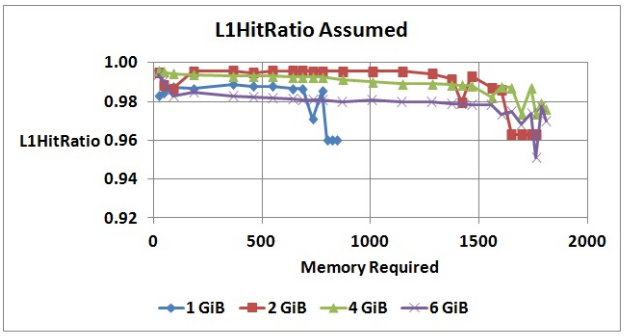


Figure 3.12: $L_1HitRatio$ chosen for Nehalem plotted against the memory required

Table 3.9: Measured Metrics from the tools

Assumed Metrics	**Calculated Metrics**
t_{hit1}	L_1Hit
t_2local	L_2Hit
t_3local	L_3Hit
$t_{mem}local$	$t_2remote$
$t_{mem}remote$	$t_3remote$
t_{disk}	RAR
	t_{hit2}
	t_{hit3}
	t_{mem}

3.3.3 Prediction and Verification

This section entails the details about the prediction of a metric on selected hardware models and the verification of the models.

The hardware models for four selected memory configurations are denoted as $\hat{Hm_1}$, $\hat{Hm_2}$, $\hat{Hm_3}$, $\hat{Hm_4}$ for 1 GiB, 2 GiB, 4 GiB and 6 GiB memory configurations respectively on machine m selected.

The metrics measured (listed in Table 3.7), and metrics assumed and calculated (listed in Table 3.9) are used to create profiles, $K1$, $K2$, $K3$, $K4$ for hardware models. The profiles are used to predict the metric, T_{tot} for all the hardware models.

$$T_{tot} = F(K1, \hat{Hm}_1)$$
$$T_{tot} = F(K2, \hat{Hm}_2)$$
$$T_{tot} = F(K3, \hat{Hm}_3)$$
$$T_{tot} = F(K4, \hat{Hm}_4)$$

The mathematical correctness of the predicted metric for every hardware model is verified by comparing it with the metric, t_{app} measured. t_{app} is the time duration between the start and the stop action of the application measured using a timer used in the MeasureLoadCLI tool. The predicted value of T_{tot} estimated using Equation 3.5 is compared with $t_{app} \notin K1, K2, K3, K4$ for all hardware models using 3.11.

$$|T_{tot} - t_{app}| \leq error \tag{3.11}$$

In order to determine if the predicted (T_{tot}) and measured (t_{app}) values of latencies are comparable, the error percentage is calculated as follows:

$$error(\%) = \frac{T_{tot} - t_{app}}{t_{app}} * 100 \tag{3.12}$$

Looking back at Figure 3.11, the step, prediction and verification are done as described in this section.

Reflection

The predicted value of the metric, T_{tot} and the error percentages calculated after comparing T_{tot} with t_{app} for all array sizes is tabulated in Tables 3.21, 3.22, 3.23, 3.24. These values depend on a number of useful observations and analysis done by comparing different memory configurations. They are discussed in detail in this section.

In Table 3.10, for $\hat{Hm}_1$,

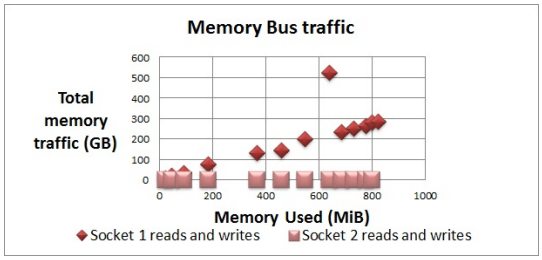
- $m.MemBw$ is calculated as (Memory Frequency $*$ Number of bytes/transfer $*$ Number of channels)=($800 * 8 * 1$=6.4 GB/s). This is the theoretical bandwidth for one socket. $m.Sock1Mem$ and $m.Sock2Mem$ indicate the amount of memory available to node 1 (socket 1) and node 2 (socket 2) respectively. These are represented separately because of NUMA.

Similarly, metrics for other hardware models are also specified in Table 3.10.

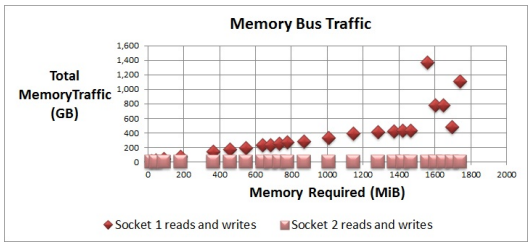
Table 3.10: Hardware models selected: The metrics are calculated from the specification sheet of the PC, and using a software, SiSoftware Sandra

Hardware Model	**Parameter**	**Value**
$\hat{Hm_1}$	m.Mem (GiB)	1
	m.MemChannel	1
	m.MemBw(GB/s)	6.4
	m.Sock1Mem(MiB)	537
	m.Sock2Mem(MiB)	0
	m.QPIBw(between sockets)(GB/s)	9.6
$\hat{Hm_2}$	m.Mem (GiB)	2
	m.MemChannel	1
	m.MemBw(GB/s)	6.4
	m.Sock1Mem(MiB)	582
	m.Sock2Mem(MiB)	861
	m.QPIBw(between sockets)(GB/s)	9.6
$\hat{Hm_3}$	m.Mem (GiB)	4
	m.MemChannel	2
	m.MemBw(GB/s)	12.8
	m.Sock1Mem(MiB)	1760
	m.Sock2Mem(MiB)	410
	m.QPIBw(between sockets)(GB/s)	9.6
$\hat{Hm_4}$	m.Mem (GiB)	6
	m.MemChannel	3
	m.MemBw(GB/s)	19.2
	m.Sock1Mem(MiB)	2130
	m.Sock2Mem(MiB)	0
	m.QPIBw(between sockets)(GB/s)	9.6

An important observation from Table 3.10 is that the memory available is always less than the total physical memory physically installed on a system. The $m.Sock2Mem$ for $\hat{Hm_1}$ is zero because no RAM card was installed on socket 2. However, $m.Sock2Mem$ for $\hat{Hm_4}$ is surprisingly found to be zero even though 3 GiB of RAM is installed on socket 2. This can be explained using the nature of the OS. A 32 bit Windows OS can use only a maximum of $2^{32} = 4GiB$ of memory. Therefore, even though $\hat{Hm_4}$ has 6 GiB of memory installed, the OS only uses the memory installed on socket 1(3 GiB). This is also proved using the measurements from the MeasureLoadCLI tool. The metrics, M_r and M_w for individual sockets are measured and plotted in Figure 3.13. It can be seen that the socket 2 does not record any traffic to the memory as it has no memory allocated.

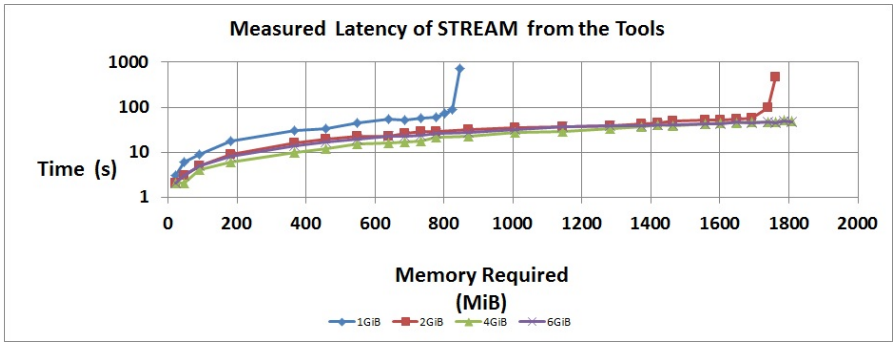


(a) Memory Bus traffic in $H\hat{m}_1$

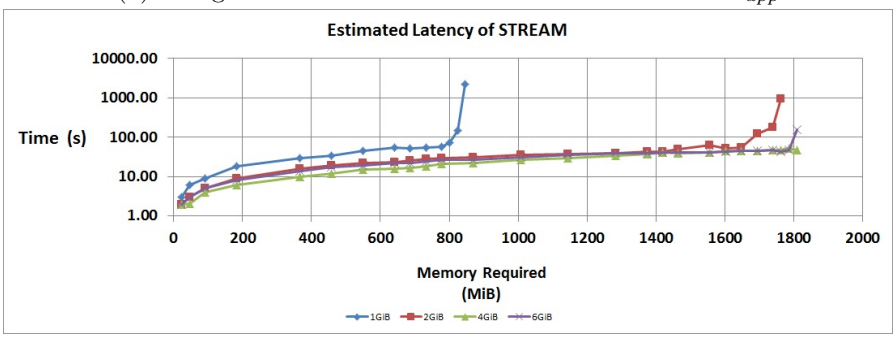


(b) Memory Bus traffic in $H\hat{m}_4$

Figure 3.13: Individual socket reads and writes in $H\hat{m}_1$ and $H\hat{m}_4$: Shows that Socket 2 does not record any memory bus traffic



(a) Using the counters in the MeasureLoadCLI tool: t_{app}



(b) Estimated (predicted) using the model:T_{tot}

Figure 3.14: Latency of STREAM application for $H\hat{m}_1$, $H\hat{m}_2$, $H\hat{m}_3$, $H\hat{m}_4$:The y-axis is the log of the values measured and predicted

t_{app} and T_{tot} tabulated in Table 3.21, Table 3.22, Table 3.23,Table 3.24 are graphically represented in Figure 3.14.In Figure 3.14a and Figure 3.14b, the latencies are plotted against the memory required for the vector kernel operations listed in Table 3.2.

From Figure 3.14, it can be seen that the estimated latency compares very well with the measured latency. There is a drastic increase in the latency of the application

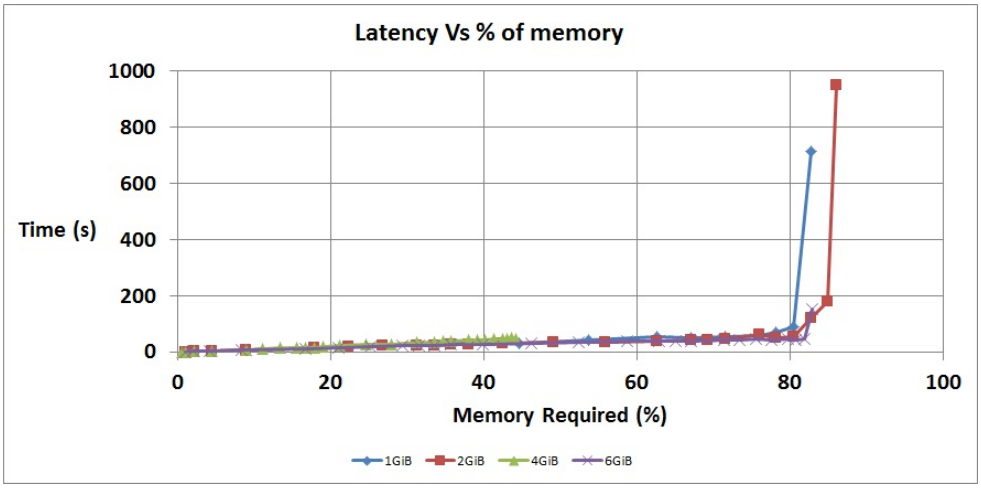


Figure 3.15: Estimated Latency of STREAM versus % of memory required

when the memory required is greater than 80% of the total memory deployed on every hardware model. This can be demonstrated using Figure 3.15. In Figure 3.15, the estimated latency of the application is plotted against the % of memory required. The % of memory required is Total memory required/Total memory as seen by the OS. For example, for $H\hat{m}_1$, the % of memory required for 1 million elements is 22.9 MiB/1 GiB=2.24%. It is seen that the percentage of memory used for the same memory requirements is higher for $H\hat{m}_4$ than that of $H\hat{m}_3$. This factor is important from the point of view of performance comparison discussed in the later section. For $H\hat{m}_4$, the total amount of memory seen by the OS is less than 3 GiB. This accounts for higher percentage of memory required for this hardware model than that required by $H\hat{m}_3$. See from Figure 3.15 that the % of memory required is never greater than 45% for $H\hat{m}_3$(4 GiB)and is greater than 80% for $H\hat{m}_4$(6 GiB) for experiments with same array size chosen.

From Table 3.21,Table 3.22,Table 3.23,Table 3.23, the rows highlighted in pink show that the modeling is valid only for those array sizes where the % of memory required is less than 80 %. The absolute value of the error percentages for valid experiments is less than 1 %. This satisfies Equation 3.11, because T_{tot} and t_{app} are comparable to each other within an absolute error of 1 %. This verifies the predictive quality of the models, $d1, \ldots d10$ for all hardware models selected. Looking back at Figure 3.11, all the steps shown are completed and the predictive quality of $d10$ is verified. $d10$ and thus, $d1, \ldots d9$ can now be formalized. The set of models can be validated using an entirely different machine, or architecture as illustrated in Figure 2.7.

3.3.4 Performance Evaluation using models

As the predictive quality of the models created are verified, we use the models to compare the performance of the PC with different memory configurations. When the memory installed on m is increased in steps of 2 GiB, we intuitively expect to see performance improvement. In terms of latency, we expect a decrease in the latency of all the experiments for every step increase in the installed memory. However, when the system was modelled using Equation 3.5, the results obtained were quite contradictory for hardware model, $\hat{Hm_4}$. This can be demonstrated using a state diagram shown in Figure 3.16. The performance improvement/degradation % is calculated as the average of the ratios of the difference between the estimated latencies of the application between two hardware models to the estimated latency of the application of one hardware model for all[8] the array sizes chosen.

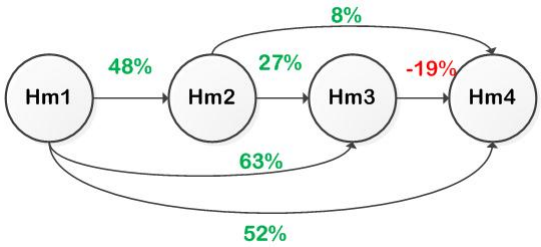


Figure 3.16: Performance Improvement in terms of Latency: The circles represent the hardware models of m; The text in green indicates performance improvement and the text in red (and minus sign) indicates performance degradation.

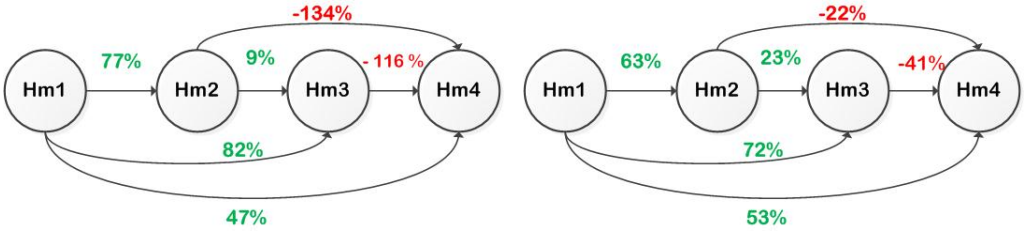
Here, if the latency estimated decreases for every step increase in the memory installed, then there is performance improvement. If the latency of the application increases, it implies performance degradation.

A performance improvement/degradation of x % from $\hat{Hm_i}$ to $\hat{Hm_j}$ means that the value of a metric kv for $\hat{Hm_j}$=(1-(x/100))$*$ kv for $\hat{Hm_i}$.

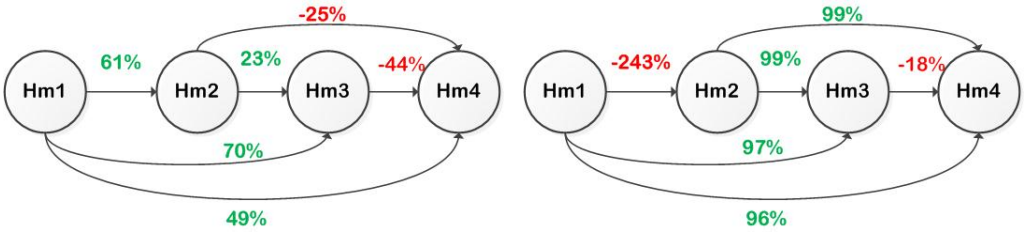
The performance improvement/degradation shown in Figure 3.16 can be explained using the metrics used in $d10 \in D$.

From Figure 3.17a, it can be seen that the number of L_1Miss decreases by 77 % from $\hat{Hm_1}$ to $\hat{Hm_2}$. The number of L_2Miss and L_3Miss also decrease by 63 % and 61 % respectively as seen in Figure 3.17b and Figure 3.17c . However, the number of disk accesses is extremely high as seen from Figure 3.17d. We still see an overall performance improvement of 48 % from $\hat{Hm_1}$ to $\hat{Hm_2}$ in Figure 3.16. An improvement of 48 % means that T_{tot} for $\hat{Hm_2}$ =0.52 $*$ T_{tot} for $\hat{Hm_1}$.

[8]All here refers to those array sizes for which the % of memory required is less than 80 %



(a) L_1Miss for hardware models (b) L_2Miss for hardware models



(c) L_3Miss for hardware models (d) $DiskHit$ for hardware models

Figure 3.17: Performance Evaluation of the estimated latencies using the model, $d10$ using the metrics: The circles represent the hardware models. Performance improvement is indicated using green text and performance degradation is indicated using red text

Table 3.11: Average of the access times to cache and memory for hardware models: The values are rounded to the nearest integers. They are the average of all the values estimated for all array sizes. The access times are not constant values, but instead are functions of RAR

Hardware Model	$thit_2$	$thit_3$	t_{mem}
$\hat{Hm_1}$	21	41	94
$\hat{Hm_2}$	12	26	74
$\hat{Hm_3}$	17	36	66
$\hat{Hm_4}$	29	43	96

Table 3.12: QPI, Remote access and Memory traffic ratios: The values are approximate and not exact values. They are the average of all the values estimated for all array sizes.

Hardware Model	$\frac{QPI_{sock2}}{QPI_{sock1}}$	**RAR**	$\frac{M_r}{M_w}$
$\hat{Hm_1}$	2.5	0.5	2.5
$\hat{Hm_2}$	1.4	0.2	2.5
$\hat{Hm_3}$	1.5	0.2	2.5
$\hat{Hm_4}$	2.5	0.5	2.5

An important inference here is that the performance improvement gained by an increase of 1 GiB in the memory configuration is not only due to the increase in the memory, but also, due to the way in which the memory is installed into the DIMMs in every

socket. In Table 3.10, and Figure 3.13, we have seen that no memory is allotted to socket 2. However, from Table 3.12, we can see that the traffic to socket 2 is 2.5 times the traffic to socket 1 and the RAR is approximately 0.5 for $H\hat{m}_1$. It is also seen that the snooping traffic is equal to the ratio of $\frac{M_r}{M_w}$ (which is 2.5). So, it can be inferred that the traffic via the QPI bus is only due to snooping. This increases the average $thit_2$, $thit_3$ and t_{mem} access times to 21 ns , 41 ns and 94 ns as shown in Table 3.11. This is higher than the local cache and memory access times discussed in Table 3.8. Refer Table 3.21 and Table 3.22 to see the access times to the cache and memory and RAR for all the array sizes. From Table 3.12, it can also be seen that for $H\hat{m}_2$, the traffic to socket 2 is 1.5 times the traffic to socket 1 via the QPI bus. The RAR is also close to 0.20. The QPI and RAR indicate that the number of remote accesses and snooping traffic is reduced. This decreases the average $thit_2$, $thit_3$ and t_{mem} access times to 12 ns, 26 ns, 74 ns from $H\hat{m}_1$ to $H\hat{m}_2$. The overall performance improvement in terms of latency can be seen in Figure 3.14.

From Figure 3.17a, it can be seen that the number of L_1Miss decreases by 9 % from $H\hat{m}_2$ to $H\hat{m}_3$. The number of L_2Miss and L_3Miss also decrease by 23 % as seen in Figure 3.17b and Figure 3.17c . The number of disk accesses is reduced by 99 % as seen from Figure 3.17d. We see an overall performance improvement of 27 % from $H\hat{m}_2$ to $H\hat{m}_3$.

The performance improvement gained from $H\hat{m}_2$ to $H\hat{m}_3$ (a step of 2GiB) is only 27 % unlike a performance improvement of 48 % from $H\hat{m}_1$ to $H\hat{m}_2$ as seen in Figure 3.16. The traffic to socket 2 is 1.5 times the traffic to socket 1 and RAR is close to 0.20 as seen from Table 3.12. The average access times, $thit_2$, $thit_3$ and t_{mem} estimated are 17 ns, 36 ns and 66 ns. The $thit_2$, $thit_3$ are higher than those estimated for $H\hat{m}_2$. Nevertheless, the number of L_1Miss, L_2Miss, L_3Miss and DiskHit are lesser in $H\hat{m}_2$. The overall performance improvement in terms of latency can be seen in Figure 3.14.

From Figure 3.17a, it can be seen that the number of L_1Miss increases by 116 % from $H\hat{m}_3$ to $H\hat{m}_4$. Similarly, the number of L_2Miss and L_3Miss also increase by 41 % and 44 % as seen in Figure 3.17b and Figure 3.17c . The number of disk accesses increases by 18 % as seen from Figure 3.17d. We see an overall performance degradation of 19 % from $H\hat{m}_3$ to $H\hat{m}_4$.

From Figure 3.17a, it can be seen that the number of L_1Miss increases by 134 % from $H\hat{m}_2$ to $H\hat{m}_4$ Similarly the number of L_2Miss and L_3Miss also increase by 22 % and 26 % as seen in Figure 3.17b and Figure 3.17c. But, the number of disk accesses reduces by 99 % as seen from Figure 3.17d. We see an overall performance improvement of 8 % from $H\hat{m}_2$ to $H\hat{m}_4$.

The performance degradation seen from $\hat{Hm_3}$ to $\hat{Hm_4}$ can be explained based on the amount of memory seen by the OS. As discussed and demonstrated earlier through Table 3.10 and Figure 3.13, the memory installed on socket 2 is not available. This is similar to $\hat{Hm_1}$ in exhibiting huge QPI traffic for snooping and leading to increased access times to cache and memory. From Table 3.11 and Table 3.12, it can be seen that the access latencies are 29ns, 43 ns and 96 ns and the RAR is approximately equal to 0.50 which are higher than the access latencies and RAR of $\hat{Hm_3}$. The overall performance degradation in terms of latency can be seen in Figure 3.14

The most interesting, yet, explanatory meagre performance improvement of 8 % from $\hat{Hm_2}$ to $\hat{Hm_4}$ is solely due to the decrease in the number of disk accesses. It can be inferred that a step of 4 GiB increase (from 2 GiB to 6 GiB) does not do the intuitive expected performance improvement. This is seen in Figure 3.14.

Note from Figure 3.16,

- The performance improvement gained from $\hat{Hm_2}$ to $\hat{Hm_4}$ is 8% which is = 27 % - 19%=8%
- The performance improvement gained from $\hat{Hm_2}$ to $\hat{Hm_3}$ is 27 % which is ≈ (48 %- 19%)=29%.

From this, it can be inferred that the performance degradation that happens when the memory is installed on only one processor(socket) or memory is available on one socket in a multiprocessor PC is ≈ 20 %.

From Figure 3.17a, it is seen that the number of L_1Miss decreases by 82 % from $\hat{Hm_1}$ to $\hat{Hm_3}$. The number of L_2Miss and L_3Miss also decrease by 72 % and 70 % as seen in Figure 3.17b and Figure 3.17c . The number of disk accesses is reduced by 97 % as seen from Figure 3.17d. We see an overall performance improvement of 63 % from $\hat{Hm_1}$ to $\hat{Hm_3}$.

From Figure 3.17a, it is seen that the number of L_1Miss decreases by 47 % from $\hat{Hm_1}$ to $\hat{Hm_4}$. The number of L_2Miss and L_3Miss also decrease by 53 % and 49 % as seen in Figure 3.17b and Figure 3.17c . The number of disk accesses is reduced by 96 % as seen from Figure 3.17d. We see an overall performance improvement of 52 % from $\hat{Hm_1}$ to $\hat{Hm_4}$.

The performance improvement gained from $\hat{Hm_1}$ to $\hat{Hm_3}$ and $\hat{Hm_1}$ to $\hat{Hm_4}$ are 63 % and 52 % respectively. The performance gain from $\hat{Hm_1}$ to $\hat{Hm_4}$ is lesser than the performance gain from $\hat{Hm_1}$ to $\hat{Hm_3}$ due to the way the memory is seen by the OS in $\hat{Hm_4}$. This is explained in earlier observations.

If $A > B$ indicates that, the performance of A is better than the performance of B, then, from Figure 3.16 and Figure 3.14, performance evaluation done can be summarized as:

$$\boxed{\hat{Hm_3} > \hat{Hm_4} > \hat{Hm_2} > \hat{Hm_1}} \tag{3.13}$$

3.3.5 Performance Evaluation using STREAM

Looking back at Figure 3.4, it was discussed that STREAM calculates the time required for vector kernel operations and the memory bandwidth. This is used to verify the results of performance evaluation done. As shown in Figure 3.4, STREAM calculates the minimum, maximum , average times and the optimum memory bandwidth for all the vector kernel operations. Note that we already have T_{tot} predicted and t_{app} measured using MeasureLoadCLI tool and the time recorded by STREAM is different from these metrics and is displayed on the command-line interface of STREAM after computations.

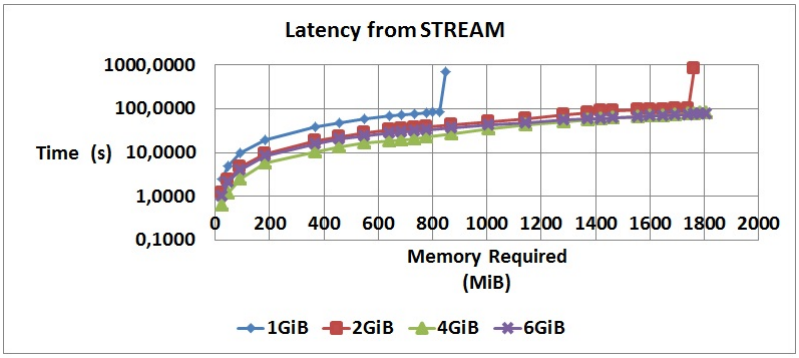


Figure 3.18: Latency recorded by STREAM application: The latency is expressed in log scale: Lower the latency, better the performance

- From Figure 3.18 it can be inferred from the latencies observed that,

$$\boxed{\hat{Hm_3} > \hat{Hm_4} > \hat{Hm_2} > \hat{Hm_1}} \tag{3.14}$$

- From Figure 3.19, it can be inferred from the memory bandwidth that,

$$\boxed{\hat{Hm_3} > \hat{Hm_4} > \hat{Hm_2} > \hat{Hm_1}} \tag{3.15}$$

Equations 3.15, 3.14 concur with Equation 3.13. This supports the performance evaluation done on different hardware models.

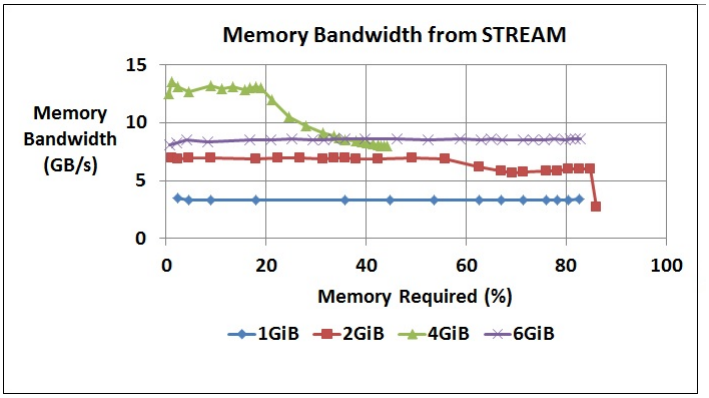


Figure 3.19: Memory Bandwidth recorded by STREAM application: The bandwidth is calculated as (Memory Required/Min Time required for vector kernel operations). Higher the bandwidth, better the performance

3.4 Execution Architecture Design-II

Intel's tick tock approach[17] moves into a new manufacturing technology every two years (called tick), the cadence Moore's law dictates. On the years in between, Intel updates the chip architecture but leaves the manufacturing process unchanged (called tock). From Figure 3.20, it can be seen that Intel roadmap has witnessed Nehalem, Sandy Bridge and Haswell micro-architectures. Each micro-architecture evolved includes enhancements in terms of power optimization, performance and many other properties from its former micro-architectures. The Allura system evolves with the Intel roadmap of processors. PCs with newer micro-architectures are integrated into the system. This motivates the comparison of PCs with different micro-architectures. Therefore, we compare the performance of two different micro-architectures, Nehalem and its successor, Sandy Bridge.

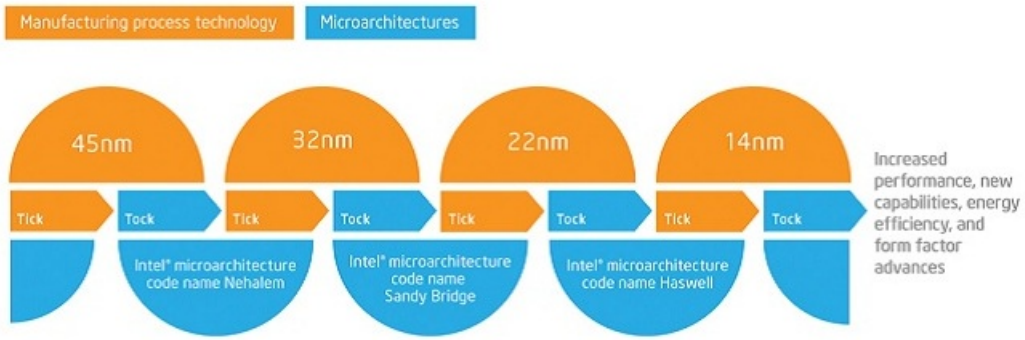


Figure 3.20: Intel Roadmap: Shows Intel's tick tock development model. Following this model, Intel commits to and delivers continued innovations in manufacturing process technology and processor architecture in alternating tick tock cycles.(Intel)

3.4.1 PC Architecture and Hardware Model

The two PCs used for study are of the type, Nehalem-EP[9] and Sandy Bridge-EP[10]. The hardware model and the physical attributes of the PC with Nehalem micro-architecture is described in Section 3.2. In this section, the hardware model and physical attributes of the PC with Sandy Bridge micro-architecture are presented.

The PC architecture consists of two CPUs running at 2 GHz. Each processor has 8 cores. Each core is connected to a private L_1 data and instruction cache of 32 KiB each and a private L_2 cache of 256 KiB. Each CPU is connected to an L_3 cache of 20 MiB which is shared among 8 cores. Every CPU supports upto 4 memory channels (bidirectional) and connects each of the 8 cores to an external memory of 32 GiB. Each Graphic card has an external memory of 2 GiB. Each socket has 40 lanes of integrated PCIe 3.0. The system is connected to 3 Graphic cards, 2 network cards and 15 grabber cards via PCIe. CPU 1 is connected to a SAS disk with a maximum capacity of 160 GiB (and 20000 rpm) via SAS LSI controller. CPU 1 is also connected to a SATA disk with a maximum capacity of 600 GiB (and 15000 rpm) via PCH. The maximum theoretical bandwidth to network card, graphic card, grabber card, SATA disk and SAS disk are tabulated in Table 3.13. This table also re-presents the hardware attributes of the Nehalem PC used for comparison. The hardware model[18] of the PC with Sandy Bridge micro-architecture chosen is shown in Figure 3.21. An instantiated CPU representation is shown in Figure 3.22.

The application used for study is STREAM. The detailed description of STREAM is already presented in Section 3.2.2

The latency of STREAM on the PC is calculated using the model developed in Section 3.2.4.The assumptions and decisions made in Section 3.2 are adopted in this section. The definition of hardware metrics used for Nehalem hold good even for Sandy Bridge. Therefore, we use the set of models ($d1, \ldots d10 \in D$) to evaluate the performance of this PC.

3.4.2 Comparison and Reflection

From the hardware models and hardware attributes described for Nehalem and Sandy Bridge, it can seen that there are many architectural differences. Nearly every aspect

[9]Intel makes two categories of processor sockets for servers: Expandable capacity (EX) and Efficient Performance(EP)processor technology. The EP processors can run on servers with a maximum capacity of 1 or 2 sockets per server. The EX processors can run on servers with 1, 2, 4, 8 , or more sockets per server

[10]Intel makes three categories of processors for Sandy Bridge: E with a maximum capacity of 1 socket (used in Desktop chips), EN with a capacity of 1 or 2 sockets per server and EP with a capacity of 1, 2, or 4 sockets per server.

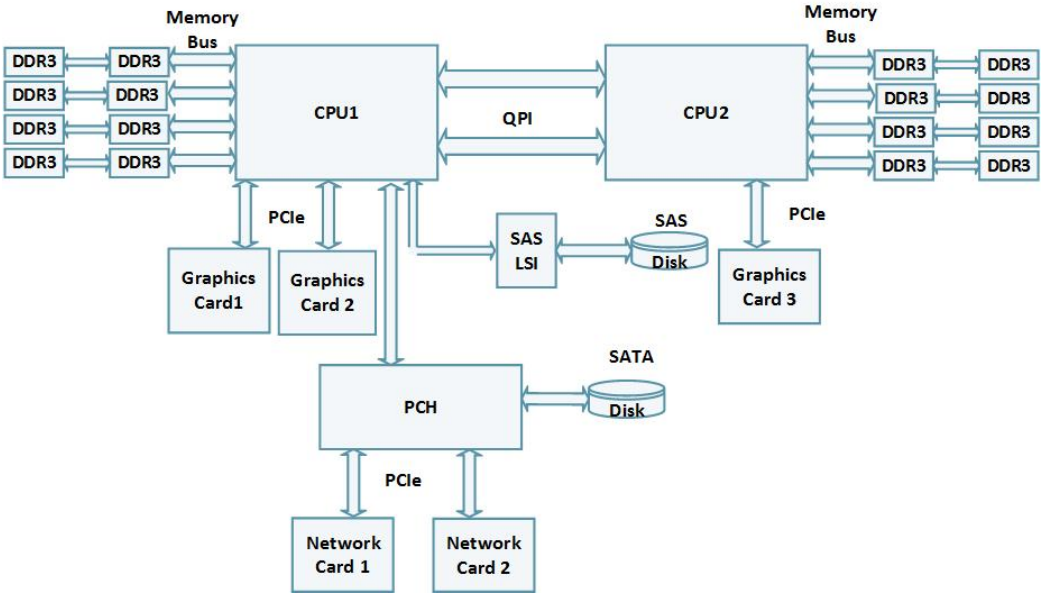


Figure 3.21: Hardware Model of the PC: Sandy Bridge

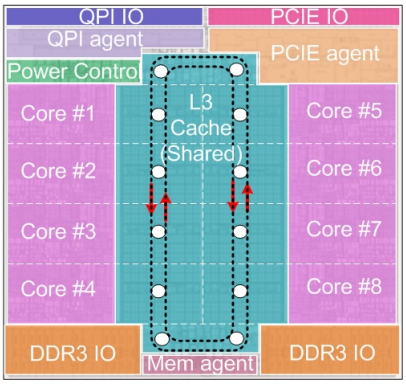


Figure 3.22: CPU: Sandy Bridge (Kanter, 2011)

of the Sandy Bridge micro-architecture has been redesigned to improve per-core performance and power efficiency[20]. It has improved CPU arithmetic, CPU multimedia, cryptography, power efficiency, media transcoding , memory controller speed and L_3 cache performances. However, this section presents major architectural advancements[21, 22, 20, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 18, 34, 35, 36] and highlights the most important architectural differences in the context of determining the performance of the PC in terms of the latency of the application chosen.

1. The Die: The major architectural changes on the die are described here. See Figure 3.22. Intel has devised innovative techniques not found on its predecessor, Nehalem. This includes:

Table 3.13: Hardware Attributes of the PCs: The hardware attributes in Table are from the device specification sheet[19, 18], BIOS settings and Si-Software Sandra.

Attribute of machine m	Sandy Bridge-EP	Nehalem-EP
$m.CPUmodel$	Intel Xeon CPU E5-2650	Intel Xeon CPU E5504
$m.Socket$	2	2
$m.CPUfreq(GHz)$	2	2
$m.Core$	16	8
$m.Thread$	1	1
$m.L_0$(KiB)	5.25	-
$m.L_1i$(KiB)	32	32
$m.L_1d$(KiB)	32	32
$m.L_2$(KiB)	256	256
$m.L_3$(MiB)	40	8
$m.MemChannel$	4	3
$m.DIMM$	2	2
$m.Mem$(GiB)	64	6
$m.MemFreq$(MHz)	1600	800
$m.TotalMemBw$(GB/s)	51.2	19.2
$m.QPIlink$	2	1
$m.TotalQPIBw$(GB/s)	32	19.6
$m.PCIeversion$	3.0	2.0
$m.Disk_{SAS}$(GiB)	160	-
$m.Disk_{SATA}$(GiB)	600	150
$m.GraphicCard$	3	2
$m.GpuMem$(MiB)	2048	512
$m.PCIeGraphicBw$ (Gbps)	40	80
$m.NetworkCard$	2	2
$m.PCIeNCBw$(Gbps)	2.5	2.5 or 10
$m.GrabberCard$	15	-
$m.PCIeGrabberBw$(Gbps)	10	-
$m.InfiniBandCard$	-	1
$m.PCIeICBw$(Gbps)	-	20
$m.TurboBoostversion$	2.0	-
$m.maxTDP$(W)	95	80
$m.lithography$(nm)	32	45

(a) Number of cores: There are 8 cores in each CPU in the Sandy Bridge PC and 4 cores in each CPU in the Nehalem PC. The number of cores was increased by Intel to be able to deliver improved performance for CPU intensive applications. As the application we chose is not CPU intensive, but hungry for memory, cache and their bandwidths, increasing the number of cores has an

adverse effect on the performance of the Sandy Bridge PC. This is explained in detail, later, in this section.

(b) L_0 cache: One of the most novel features of the Sandy Bridge micro-architecture is that, it includes a level 0 cache. This is also known as the micro-op cache. It is a subset of the L_1 instruction cache that contains fixed length decoded micro-ops, rather than raw bytes of variable length instructions. The key aims of this cache are to improve the performance of CPU intensive applications by improving the bandwidth available at the front-end of the CPU and remove decoding from the critical path. It also decreases the power consumption.

The size of L_0 cache is 5.25 KiB. The L_0 has a hit rate of about 80 %.

(c) L_1 cache: The L_1 data cache in Sandy Bridge micro-architecture is redesigned to increase the bandwidth. It can sustain two 128 bit loads and one 128 bit store every cycle, unlike Nehalem which can sustain only one 128 bit load and one 128 bit store. This doubles the the bandwidth available to the cache directly connected to the load and store buffers, L_1 cache.

(d) L_3 cache:

i. Location: Unlike the L_3 cache located in the un-core[11] domain in the Nehalem PC, L_3 cache is directly connected to the cores in the processor clock domain on the Sandy Bridge PC.

ii. Capacity: The capacity of L_3 cache is 20 MiB which is 2.5 times the capacity of L_3 cache in the Nehalem PC. Another important difference between the two L_3 caches is that in Sandy Bridge micro-architecture, the L_3 cache is divided into 8 equally sized physical blocks(logically, there is one L_3 cache). This allows each block to service requests simultaneously making its bandwidth approximately 8 times larger.

(e) System Agent: Sandy Bridge contains a new component, the System Agent, that controls the memory controller, PCI Express, display interfaces,and the DMI connection to the external south-bridge chip (PCH). The un-core domain in the Nehalem micro-architecture is replaced by the system agent. The effect of System Agent is not significant in this context.

(f) Ring Interconnect:The Sandy Bridge micro-architecture employs a ring interconnect between the cores, graphics, L_3 cache and system agent (including the display/media engine) which replaces the internal buses used in the Nehalem

[11]The uncore is a term used by Intel to describe the functions of a microprocessor that are not in the Core, but which are essential for Core performance, such as, L_3 cache, power control unit, on die memory controller, QPI controller. Un-core frequency is lower than the core frequency

micro-architecture. The coherent bidirectional ring is composed of four different rings: request, snoop, acknowledge and a 32 B wide data ring. Together these four rings are responsible for a distributed communication protocol that enforces coherency and ordering. The rings are fully *pipelined* and run at the core clock and voltage. The L_3 cache and the data rings in the system fabric run at the core frequency providing upto 844 GB/s of fabric bandwidth. However, the L_3 cache in Sandy Bridge is shared by the cores, the integrated GPU and the system agent. Each of these agents accesses the L_3 cache via the ring. The bandwidth scales with the agents. But, the scaling is not necessarily perfect though, because of the topology. As messages travel across the ring, they can *block access* to other agents, reducing the available bandwidth as the average hop count (e.g, number of cores) increases. Therefore, L_3 cache is no longer a single unified entity as in Nehalem, but is instead distributed and partitioned for higher bandwidth and associativity.

From the factors mentioned, it can be summarised that, first, each slice of the L_3 (20 MiB/8=2.5 MiB)cache in the Sandy Bridge micro-architecture is much smaller (≈ 3 times) than Nehalem's 8 MiB L_3 cache, so the latency to access the tags and data arrays has decreased. Second, the ring and L_3 now reside in the same clock and voltage domain as the cores (and the core clock is certainly faster than the un-core clock in Nehalem). In Nehalem micro-architecture, there is a latency penalty for signals crossing to a new voltage and clock domain. This penalty is determined by the ratio between the two frequencies and can be several cycles. Placing the caches and ring in the same domain reduces this latency.

Note that the latency of the ring will increase as more agents are attached; each hop on the ring takes 1 cycle, so the latency actually depends on the relative position of the requesting core and the receiving cache slice. The bandwidth available also depends on the blocking caused due to other agents. Nevertheless, the access time to L_3 cache is reduced by 25 % (40 clock cycles in Nehalem to 30 clock cycles in Sandy Bridge).

(g) GPU on the die: GPU is directly connected to the die in the Sandy Bridge PC. This improves the speed of the components used in GPU. The GPU functionalities are enhanced and redesigned for better performance. As this is beyond the scope of this thesis, it is not discussed in detail.

(h) Other changes: As discussed earlier, every aspect of Sandy Bridge PC is redesigned to improve the performance. Some of the changes are: improved branch prediction, increased number of load and store buffers, efficient techniques for tracking and renaming micro-ops in flight, introduction of the AVX instruction set for improved floating point performance. As these finer details cannot be measured using the tools we use, they are not discussed in detail.

2. Memory: The memory outlay, frequency and configuration in Sandy Bridge are the most important factors affecting the performance in this context.

   (a) Frequency: The memory frequency doubles (from 800 MHz in Nehalem to 1600 MHz in Sandy Bridge) improving the access time to memory by 100 %.

   (b) Memory Channels: The number of memory channels increases (maximum of 3 channels in Nehalem to 4 channels in Sandy Bridge), thereby increasing the memory bandwidth by 33 % . However, we consider the best possible configuration supported on the Nehalem PC with 4 GiB of memory configured and 2 memory channels used for our study. Therefore the memory bandwidth available on a Sandy Bridge PC is twice of that available on the Nehalem PC.

   (c) Capacity: The maximum memory configuration used in the Nehalem PC is 6 GiB. On the other hand, Sandy Bridge has a total of 64 GiB of memory installed in the system. This makes the number of disk accesses very low.

3. QPI links:

   (a) Number of QPI links: The number of QPI links between the sockets is doubled (From 1 bidirectional link in Nehalem to 2 bidirectional link in Sandy Bridge). The number of QPI links in Sandy Bridge micro-architecture is increased to ensure scalability in servers that support many sockets. However, the PC we used for study supports two sockets only. Also, from the experiments conducted on the Nehalem PC, it was seen that the % of QPI bandwidth used ranges from 10 % to 30 %. Therefore, we ignore the effect of increased number of QPI links in our study.

   (b) QPI bandwidth : The bandwidth of each QPI link in the Sandy Bridge PC is 16 GB/s which is approximately 1.6 times the QPI bandwidth supported in the Nehalem PC (9.8 GB/s). This plays a significant role in improving the performance of the system.

   (c) QPI protocol: QPI 1.1 used in Sandy Bridge is different from QPI 1.0 link used in Nehalem, both, on the physical organisation level and the protocol implementation. The primary change is that QPI 1.1 uses home based snooping technique unlike Nehalem which uses source based snooping technique. In source snooping, the requesting processor that missed in the L_3 cache broadcasts a snoop request to the entire system. Other caching agents (i.e., anything with a cache, such as another processor) may fulfil the snoop request if they hold a cached copy of the data. The home agent (i.e., the memory controller that owns the data) will respond to the snoop with a clean copy of the cache line if necessary. The home node still receives all of the acknowledgements from the caching agents and if a conflict occurs, will resolve

transactions in the correct order. In home snooping, the requesting processor sends a request to the home agent. Second, the home agent will send snoop requests only to the caching agents that have a copy of the data (filtering) and possibly begin reading the cache line from memory. Lastly, the home node and (or) any caching agents will send data to the original requester.

The coherency management resides with the home agent in home snooping, which makes it simpler. Source snooping results in lower latency, especially when the requested cache line is held in remote memory and remote cache. However, home snooping is a more natural fit for inter-socket snoop filtering in servers with many sockets. We focus on two socket systems only. Therefore, we ignore the benefits of home snooping protocol.

4. IO Hubs: One of the most notable, and interesting changes in the hardware models of the two types of PCs presented is the removal of IO Hubs in the Sandy Bridge micro-architecture. The PCIe slots are directly connected to the die providing faster access and improved scalability. The system agent provides a direct connection from the CPU1 to the PCH that connects to the SATA disks, USB devices, network cards and grabber cards. A direct interface from CPU1 to SAS ports is supported by SAS LSI.

   These features are remarkably different in Nehalem micro-architecture, where, the IO Hubs connect the PCIe slots, network, graphic cards to the CPUs. An ICH is connected to the IO Hub 1 which provides interface to the SATA disks used. The QPI links are used to connect the CPUs to the IO hubs.

   Though the changes are notable, they do not contribute to performance improvement in the context of this project because the application we chose does not exercise the network card, graphic card and their links. As the memory installed is 64 GiB and the OS used is Windows 7 (64 bit), all of the memory is available to the OS and accounts for no disk accesses.

5. PCIe: The Sandy Bridge PC integrates PCI-Express 3.0 unlike Nehalem which uses PCI-Express 2.0. PCIe 3.0 provides higher bandwidth and better data encoding techniques. The application we chose does not exercise the PCIe. Therefore, we do not delve into further details.

6. Turbo Boost [12]: The Sandy Bridge PC we used for study uses Turbo Boost 2.0 as against the Nehalem PC which does not use this technology. In the Sandy Bridge PC, the processor and its thermal system cools down when the CPU is idle. It uses these thermal reserves and dynamically ramps the frequency and the supply

[12]Intel Turbo Boost is a technology implemented by Intel in certain processors that enables the processor to run above its base operating frequency via dynamic control of the CPU's clock rate. It is activated when the operating system requests the highest performance state of the processor.

voltage to operate above its rated TDP[13](for about 25 seconds according to Intel) until the thermal system heats up again in a situation dependent manner.

Note that the Turbo Boost feature can be disabled / enabled using the BIOS settings in a PC. In our study, the Turbo Boost technology is enabled on the Sandy Bridge PC.

The hardware attributes of both the PCs are listed in Table 3.13. It entails the similarities and differences between the two PCs using numerical values.

To summarize this, the differences observed and the performance improvement or degradation expected (when using STREAM) due to these differences in the two PCs are presented in Table 3.14, Table 3.15, Table 3.16, Table 3.17.

Table 3.14: Effect on the Performance due to L_1 cache

Changing Factor	**Nehalem**	**Sandy Bridge**	**Effect on the Performance of Sandy Bridge**
Bandwidth	1 load and 1 store	2 load and 1 store	Bandwidth improvement
Total Effect			$t_{hit1} = \frac{1}{2} * t_{hit1}$ for Nehalem $= \frac{2ns}{2} = 1$ ns.

Table 3.15: Effect on the Performance due to L_3 cache

Changing Factor	**Nehalem**	**Sandy Bridge**	**Effect on the Performance of Sandy Bridge**
L_3 cache Capacity per core(MiB)	8	2.5	
Location	In the un-core domain	In the processor clock domain	
Connection to other agents	Internal Bus	Ring Interconnect in the processor clock domain	
Total Effect			t_3local reduces by 25 %. $\therefore t_3local = t_3local$ in Nehalem -(t_3local in Nehalem)/4= 20-(20/4)=15 ns

Table 3.16: Effect on the Performance due to Memory

Changing Factor	**Nehalem**	**Sandy Bridge**	**Effect on the Performance of Sandy Bridge**
Memory Frequency(MHz)	800	1600	t_{mem} reduces
Memory Channels	2	4	t_{mem} reduces
Total Effect			$\frac{(m.MemChannels/m.Cores) * m.MemFreqforSandyBridge}{m.MemChannels/m.Cores * m.MemFreqforNehalem} = \frac{(4/8) * 1600}{(2/4) * 800} = 2.$ $\therefore$, $t_{mem}local = \frac{1}{2} * t_{mem}local$ in Nehalem . $t_{mem}local = \frac{65}{2} = 32.5$ ns

[13]The thermal design power (TDP), sometimes called thermal design point, refers to the maximum amount of power the cooling system in a computer is required to dissipate. It is typically the maximum power that it would draw when running real applications. The lower it is, the better.

Table 3.17: Effect on the Performance due to QPI

Changing Factor	Nehalem	Sandy Bridge	Effect on the Performance of Sandy Bridge
QPI Bandwidth per link(GB/s)	9.8	16	Bandwidth increases
Total Effect			$\frac{m.QPIBwforSandyBridge}{m.QPIBwforNehalem} = \frac{16}{9.8}$ =1.63. ∴ QPI latency reduces to $\frac{40ns}{1.63}$ =25 ns .

3.5 Experiments and Results-II

This section gives a detailed description of the experiments conducted on the Sandy Bridge PC. It describes *Series of experiments done to compare Nehalem and Sandy Bridge micro-architectures.* In Section 3.3, the experiments done on the Nehalem PC are presented. The models, $d1, \ldots d10$, notations and the experimental results presented in Table 3.21, Table 3.22, Table 3.23, Table 3.24 are used in this section for comparison.

The series of steps illustrated in Figure 3.10 are adopted in this section. The hardware specifications of the PC on which the experiments are conducted are tabulated in Table 3.13 and the parameters chosen for STREAM are presented in Table 3.6. The procedure of conducting experiments are adopted from Section 3.3.1. The OS used on the Sandy Bridge PC is Windows 7. The list of measured metrics are also the same on the Sandy Bridge PC as listed in Table 3.7, except for QPI_{sock}. The QPI traffic between the sockets cannot be measured on the Sandy Bridge PC using the Intel PCM.

3.5.1 Assumptions and Calculations

The effect on the metrics due to the change in the micro-architecture that are relevant to the model presented in Table 3.14, Table 3.15,Table 3.16, Table 3.17 are used to calculate the local and remote access times to L_1, L_2, L_3 caches, memory and disk. This is tabulated in Table 3.18. This in turn, is used to calculate the total access times to the caches and memory for different array sizes. All these are tabulated in Table 3.20.

The hardware model of the PC with Sandy Bridge micro-architecture is denoted as $\hat{Hm}_5$. The metric, RAR represented by $d1$ in Section 3.2.4 is assumed to be 0.20. This assumption is required because the Intel PCM does not capture QPI metrics on the PCs with Sandy Bridge micro-architecture. However, the total memory is symmetrically distributed between the two sockets and the OS used is Windows 7 (64 bit). In Section 3.3.3 and Table 3.12, we have seen that for hardware models with symmetric memory configuration on both sockets, the average RAR is approximately 0.20. This is demonstrated using Figure 3.23. It can be seen that for $\hat{Hm}_2$ and $\hat{Hm}_3$ the RAR is approximately 0.20 and for $\hat{Hm}_1$ and $\hat{Hm}_4$ (asymmetric memory configuration), the value

Table 3.18: Access Times to Cache and Memory

Cache Level /Memory	Local	Remote
$L_1(ns)$	t_{hit1}	t_{hit1}
	1	1
L_2(ns)	t_2local	$t_2remote$
	6	6+25=31
L_3(ns)	t_3local	$t_3remote$
	15	15+25=40
DRAM(ns)	$t_{mem}local$	$t_{mem}remote$
	32.5	32.5 + 25=57.5
	t_{disk}	t_{disk}
Disk(ms)	2	2

of RAR is approximately 0.50. Intel also uses this principle to define various metrics in PCM[37]. Therefore, we chose RAR=0.20 for all experiments conducted on the PC with Sandy Bridge micro-architecture. This value remains constant for all array sizes.

The $L_1hitRatio$ cannot be measured using the tools selected (same like Nehalem). Therefore, we chose $L_1HitRatio$=0.94 for the experiments. The values chosen for $L_1HitRatio$ gives the best regression fit for every experiment. All these are tabulated in Table 3.20. Looking back at Figure 3.10, $L_1HitRatio$ is a part of the curve fitting.

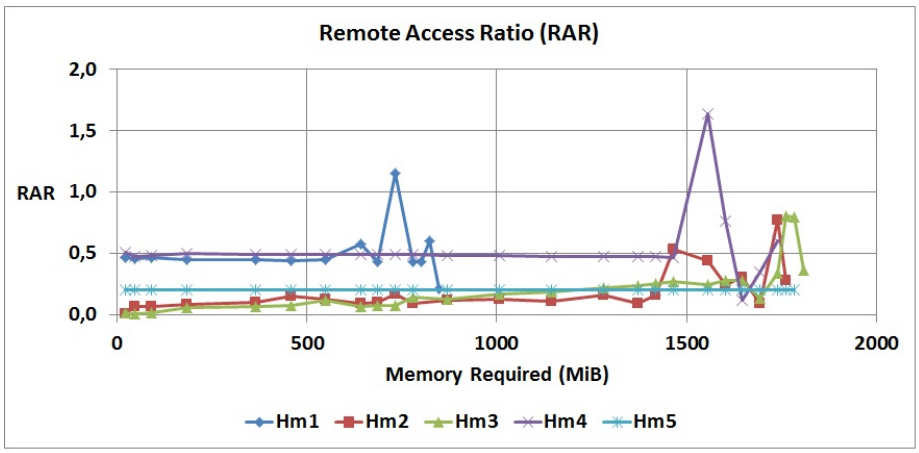


Figure 3.23: Remote Access Ratio for Nehalem and Sandy Bridge micro-architecture: The RAR of all the hardware models of the PC with Nehalem micro-architecture and the PC with Sandy Bridge micro-architecture

The metrics assumed and calculated are summarised in Table 3.19.

Table 3.19: Measured Metrics from the tools

Assumed Metrics	**Calculated Metrics**
t_2local	t_{hit1}
RAR	L_1Hit
t_{disk}	t_3local
	L_2Hit
	L_3Hit
	$t_{mem}local$
	$t_{mem}remote$
	$t_2remote$
	$t_3remote$
	t_{hit2}
	t_{hit3}
	t_{mem}

3.5.2 Prediction and Verification

From the metrics captured using the tools, metrics assumed and metrics calculated, a profile, $K5$ is created for $H\hat{m}_5$. This is used to predict a metric, T_{tot} for all the experiments. T_{tot} is predicted as follows:

$$T_{tot} = F(K5, H\hat{m}_5)$$

The estimated (predicted) latency, T_{tot} is verified by using $t_{app} \notin K5$. This is done as follows.

$$|T_{tot} - t_{app}| \leq error \tag{3.16}$$

In order to determine if the predicted (T_{tot}) and measured (t_{app}) values of latencies are comparable, the error percentage is calculated as follows:

$$error(\%) = \frac{T_{tot} - t_{app}}{t_{app}} * 100 \tag{3.17}$$

From Table 3.20, it is seen that T_{tot} is compared with t_{app}. The average absolute value of the error percentage of all array sizes is 15 %. This verifies the predictive quality of $d1, \ldots d10$ for $H\hat{m}_5$ and subsequently, verifies the choice of access times and RAR metrics assumed.

From Figure 3.24, it is seen that the measured and estimated latencies are coherent with respect to each other for all array sizes chosen during experiments.

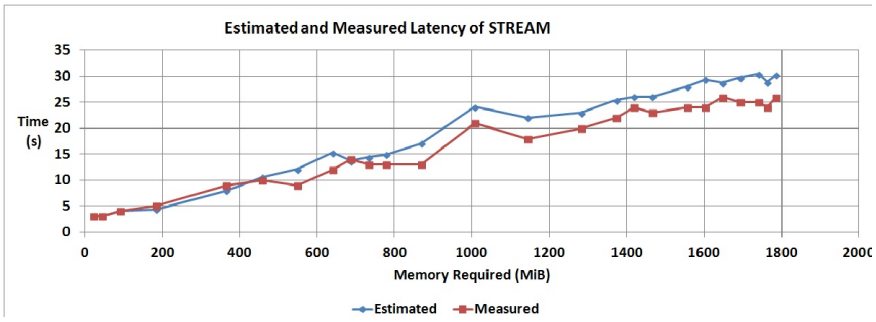


Figure 3.24: Estimated and Measured Latency of STREAM for Sandy Bridge micro-architecture: Estimated latency is the latency predicted using the models. Measured latency is from the tools.

From Figure 3.10, models $d1, \ldots d10$ can be formalized. Also, Looking back at Figure 2.7, the models are validated on a machine with entirely different micro-architecture.

Although the latencies predicted and measured are comparable, they do not follow a smooth path, but instead show noticeable variations. On increasing the array size, the latency of the application is expected to increase. However, from Figure 3.24, it can be seen that, not all array sizes show this behaviour. This can be explained using the *Active Relative Frequency (ARF)* metric for the system captured using the MeasureLoadCLI tool. ARF is defined as the ratio of the frequency at which the system runs to the base clock frequency of the system. An ARF value greater than 1 indicates that the Turbo Boost technology is enabled on the system. In Figure 3.25,the estimated latency of the application for all experiments conducted is plotted against the ARF of all the processors captured (expressed as percentage). It can be seen that the ARF varies from 110 % to 119 %. The green blocks indicate performance boost achieved due to the ramping up of the processors' frequency. *Performance boost* in this context means that the latency of the application does not increase when the array size is increased. It remains constant or decreases with respect to its former experiments (where smaller array sizes were chosen). From the first green box, it can be seen that the latency estimated does not increase when the array size is increased from 1 million elements to 2 million elements, but instead remains to be 3 s. This is because the ARF changes from 110 % to 114 %. The ramping up of the frequency of the processors in the system accounts for this performance boost observed. With increased frequency, processors are faster and this explains lower latencies for higher array sizes[14]. Similarly, note that for every occurrence of performance boost, there is a corresponding increase in the ARF.

In STREAM, the highest array size that can be chosen is about 80 million elements. It is a memory bound application

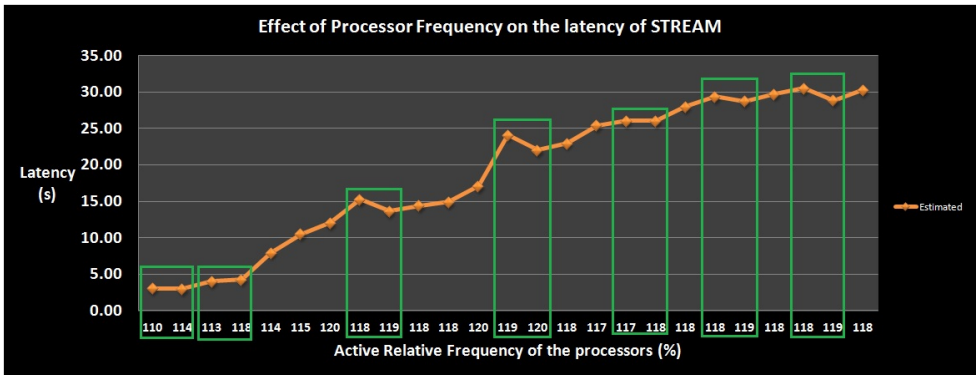


Figure 3.25: Effect of Processor Frequency on the performance of STREAM application

3.5.3 Performance Evaluation using models

The performance of the hardware model with Sandy Bridge micro-architecture, $H\hat{m}_5$ is compared with all the hardware models with Nehalem micro-architecture. From Figure 3.26, it is seen that the latency of STREAM estimated on the PC with Sandy Bridge micro-architecture is lower than the latencies estimated for all hardware models of the PC with Nehalem micro-architecture. Sandy Bridge definitely exhibits higher performance than Nehalem.

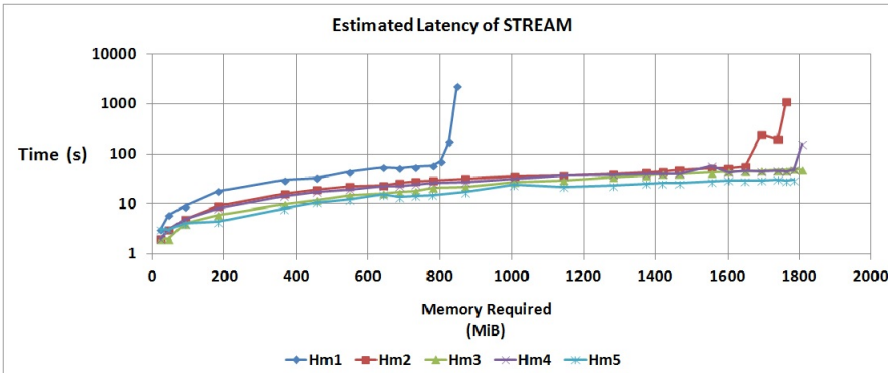


Figure 3.26: Estimated Latency of STREAM for all hardware models of Nehalem and Sandy Bridge micro-architecture: The estimated latency is plotted against the memory required for experiments with different array sizes

The performance gain of $H\hat{m}_5$ is estimated with respect to $H\hat{m}_1$, $H\hat{m}_2$, $H\hat{m}_3$, $H\hat{m}_4$ and shown using a state diagram representation.

From Figure 3.27, it is seen that the PC with Sandy Bridge micro-architecture shows a performance improvement of 26 % from the hardware model that exhibits the best performance on the PC with Nehalem micro-architecture, $H\hat{m}_3$(4GiB memory configu-

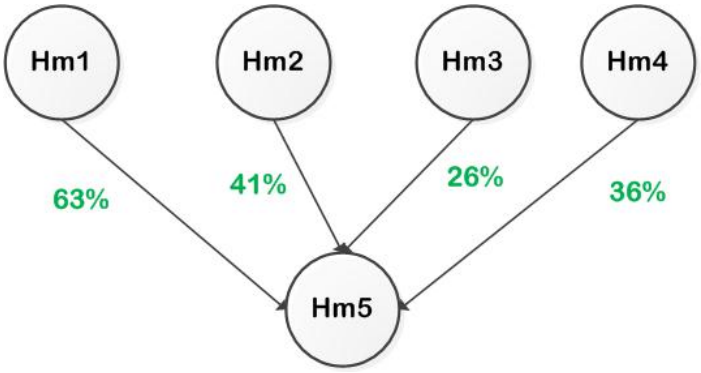


Figure 3.27: Performance Evaluation: The average value of ratios of the difference between the latencies of two micro-architectures to the latency of one micro-architecture for all array sizes is shown. The text highlighted in green indicates performance improvement.

ration). It shows a performance improvement of 36 % from the hardware model, $\hat{Hm_4}$ (6 GiB), a performance improvement of 41 % from $\hat{Hm_2}$ and 63 % from $\hat{Hm_1}$. From equation 3.13, we already expect that the performance improvement of $\hat{Hm_5}$ is the highest with respect to $\hat{Hm_1}$ and then followed by $\hat{Hm_2}$, $\hat{Hm_4}$ and $\hat{Hm_3}$ in the same order as stated. This can also be seen in Figure 3.26.

From Figure 3.26 and Figure 3.27, it can be inferred that :

$$\boxed{\hat{Hm_5} > \hat{Hm_3} > \hat{Hm_4} > \hat{Hm_2} > \hat{Hm_1}} \tag{3.18}$$

3.5.4 Performance Evaluation using STREAM

The latency of the vector kernel operations and the memory bandwidth recorded by STREAM shown in Figure 3.28 and Figure 3.29 is used to support the performance evaluation done on the Sandy Bridge and Nehalem PCs. This is done as discussed in Section 3.3.5.

From Figure 3.28, the performance of the all hardware models can be summarized as:

$$\boxed{\hat{Hm_5} > \hat{Hm_3} > \hat{Hm_4} > \hat{Hm_2} > \hat{Hm_1}} \tag{3.19}$$

From Figure 3.29, the performance of the PCs for all hardware models in terms of memory bandwidth can be summarized as:

$$\boxed{\hat{Hm_5} > \hat{Hm_3} > \hat{Hm_4} > \hat{Hm_2} > \hat{Hm_1}} \tag{3.20}$$

The memory bandwidth recorded for $\hat{Hm_5}$ is twice as much the memory bandwidth recorded for $\hat{Hm_3}$. This can be explained using the analysis done in Table 3.16. The

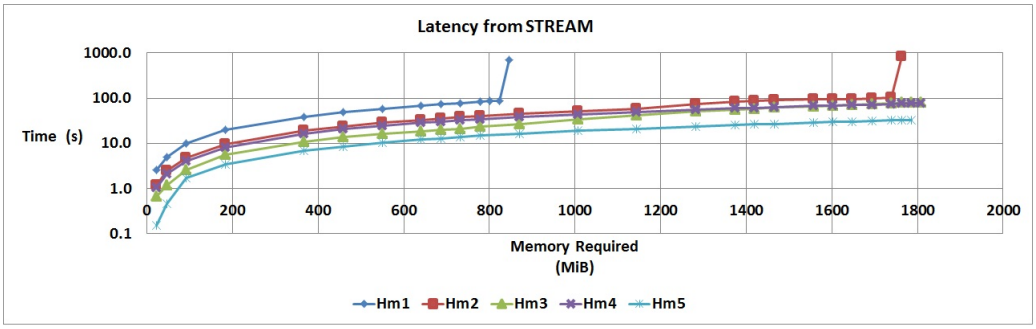


Figure 3.28: Latency of the vector kernel operations recorded by STREAM: The latency for all hardware models selected are plotted against the memory required for different array sizes chosen.

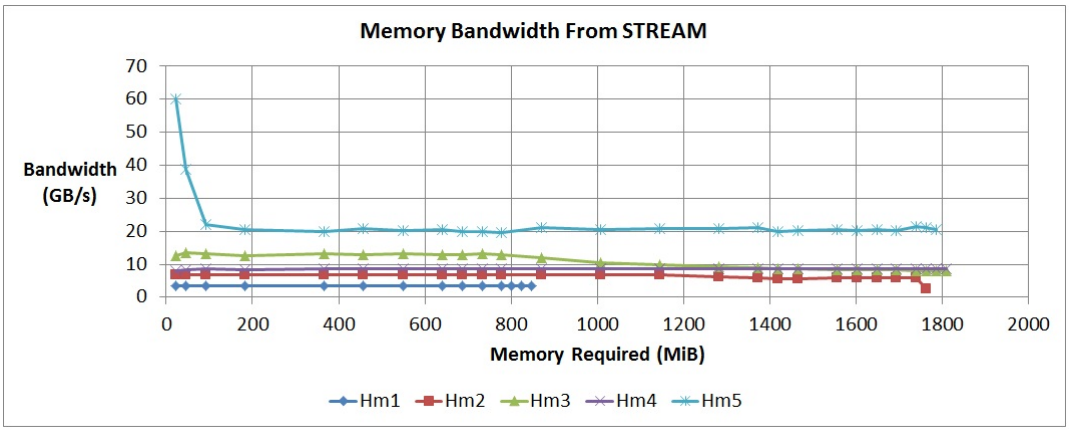


Figure 3.29: The Memory Bandwidth for vector kernel operations in STREAM: The total bandwidth is calculated according to the method shown in Figure 3.4. Memory Bandwidth for various array sizes chosen is shown.

peaks seen for the first few experiments in $H\hat{m}_5$ is due to the size and bandwidth of L_3 cache on the PC with Sandy Bridge micro-architecture. The total size of the L_3 cache is 40 MiB. For those experiments that do not meet the general rule of STREAM discussed in Section 3.2.2, array size chosen is small and fits in the L_3 cache and the memory bandwidth shown in Figure 3.29 also includes the bandwidth contributed by L_3 cache. Once the array size chosen does not fit in the L_3 cache, the memory bandwidth stabilizes.

Equation 3.18 derived from modeling concurs with Equation 3.20 and Equation 3.19 derived from STREAM.

From the analysis done so far, for the PCs chosen, and models created, Sandy Bridge micro-architecture out-performs Nehalem micro-architecture.

3.6 Lessons Learned

This section lists some of the most important observations made when creating the models. They are:

1. Effect of Interleaved memory access: When the number of memory channels are increased in the architecture of a PC, the performance increases because of interleaved memory access principle used in accessing the data from the memory. In interleaved memory, memory addresses are allotted to alternate memory banks. For example, in a processor with 4 memory channels, the first 64 bytes is allotted to memory bank connected to channel 1, the next 64 bytes is allotted to the memory bank connected to channel 2 and so on. Memory access, which according to the locality principle mainly happens in adjacent memory areas, is thus distributed across all channels. When the data is accessed in parallel through all the memory channels, there is a definite performance gain. Therefore, the memory bandwidth increases and latency decreases with the increase in the number of memory channels.

2. Verification of models: The models, $d1, \ldots d10$ created by using the Nehalem micro-architecture is verified using a measured metric not used in the model. The error percentage calculated between the predicted and measured metric is less than 1 % for the cases when the model is valid.

3. Cost Reduction: From the performance evaluation done on the PC with Nehalem micro-architecture, if the OS used is Windows XP, then the performance of the PC with 4 GiB memory is better than the performance of the same PC with 6 GiB of memory. Unused memory can be removed from the PC to achieve cost reduction.

4. Validation of models: The models, $d1, \ldots d10$ are validated by testing the predictive quality of the models on an entirely different machine, with a different micro-architecture. The average value of the absolute error percentages for all array sizes in the experiments is 15 %. It is also to be noted that the machines have to be physically available for the purpose of validation for the models created. Being able to predict the performance of the machines which are not available, but whose specifications are known is the next step. This is future work.

5. Usefulness of STREAM: The STREAM benchmark program generates huge amount of traffic on the memory buses. Though the ultimate aim of modeling is being able to use it in real time applications, STREAM represents memory intensive operations. As a next step, the models created can be tested on a machine with real time memory intensive operations.

Table 3.20: Experimental results of $\hat{Hm}_5$: Sandy Bridge micro-architecture

N(millions) / Metrics	1	2	4	8	16	32	56	60	62	64	68	70	72	74	76	77	78
$L_2HitRatio$	0.3980	0.4333	0.2642	0.2213	0.2148	0.2442	0.2233	0.2064	0.21323836	0.22961775	0.2064	0.1984	0.2086	0.2235	0.2071	0.2165	0.2171
$L_3HitRatio$	0.4356	0.3289	0.1823	0.0951	0.1139	0.1063	0.0614	0.0721	0.0863	0.0727	0.0724	0.0702	0.0807	0.0778	0.0756	0.0541	0.07880
$L_2Miss(10^6)$	4.3	13.7	45.5	70.8	129.9	229.8	377.6	425.9	432.2	428.8	470.4	497.6	478.5	493.0	512.7	482.2	502.8
$L_3Miss(10^6)$	1.9	11.4	42.4	67.3	124.7	222.0	368.4	408.5	415.53	412.35	449.4	473.5	460.9	471.0	488.3	463.6	482.5
$L_1HitRatio$	0.9976	0.9898	0.9710	0.9400	0.9400	0.9400	0.9400	0.9400	0.9400	0.9400	0.9400	0.9400	0.9400	0.9400	0.9400	0.9400	0.9400
$DiskHit$	1	0	1	0	1	0	0	2	0	0	0	1	0	0	3	0	1
$MemHit(10^6)$	1.9	11.4	42.4	67.3	124.7	222.0	368.4	408.5	415.5	412.4	449.4	473.5	460.9	471.0	488.3	463.6	482.5
$L_3Hit(10^6)$	1.5	5.6	9.4	7.1	16.0	26.4	24.1	31.7	39.2	32.3	35.1	35.8	40.5	39.7	39.9	26.5	41.3
$L_2Hit(10^6)$	2.8	10.5	16.3	20.1	35.5	74.3	108.6	110.8	117.2	127.8	122.4	123.2	126.1	141.9	133.9	133.2	139.4
$L_1Hit(10^6)$	2900	2400	2100	1400	2600	4800	4900	7300	7600	8400	8600	8700	9300	9700	9500	9900	10100
$RemoteAccessRatio$	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
$t_{hit2}(ns)$	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0
$t_{hit3}(ns)$	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
$t_{mem}(ns)$	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5
$T_{tot}(s)$ 3.08	3.01	4.03	4.31	7.98	14.44	14.91	22.99	25.43	26.07	26.07	28.01	29.38	28.75	29.77	30.52	28.89	30.31
$t_{app}(s)$	3	3	4	5	9	13	20	22	24	23	24	24	26	25	25	24	26
% of Memory Required	0.03	0.07	0.14	0.28	0.56	1.12	1.96	2.10	2.17	2.24	2.37	2.44	2.51	2.58	2.65	2.69	2.72
$error(\%)$ -2.58	-0.22	-0.75	13.76	11.34	-11.05	-14.66	-14.93	-15.57	-8.61	-13.37	-16.71	-22.40	-10.57	-19.07	-22.08	-20.37	-16.58

Table 3.21: Experimental results with 1 GiB Memory Configuration ($\hat{Hm}_1$): Nehalem micro-architecture

N(millions) / Metrics	1	2	4	8	16	20	24	28	30	32	34	35	36	37
$L_2HitRatio$	0.4273	0.4641	0.5026	0.4787	0.5099	0.5148	0.5163	0.5179	0.5219	0.5300	0.5251	0.4735	0.4368	0.3110
$L_3HitRatio$	0.1803	0.1799	0.1445	0.1325	0.1072	0.1099	0.1035	0.1083	0.1136	0.1158	0.1103	0.1339	0.1482	0.1956
$L_2Miss(10^6)$	9.3	17.2	21.0	46.3	62.5	74.1	100.2	126.8	125.1	172.0	144.3	191.1	293.7	1800
$L_3Miss(10^6)$	7.7	14.0	17.9	39.7	55.1	64.8	88	111	108.4	150.5	124	159.4	247.1	1400
$Mreads + Mwrites(10^9B)$	9.2	22.1	36.1	77.4	134.7	148	198.8	245.3	235	339.4	262.5	280.1	491	1100
$QPI_{socket}(10^9B)$	4.3	10	16.9	34.6	60.1	65.7	88.6	141.1	102.6	392.3	114.3	122	297.1	237.1
$L_1HitRatio$	0.9830	0.9845	0.9872	0.9866	0.9889	0.9879	0.9877	0.9865	0.9865	0.9710	0.9854	0.9600	0.9600	0.9600
$DiskHit$	103.4	104.1	102.8	104.2	197.9	106.5	98.7	95.4	94.8	95.1	1100	17600	58400	1100000
$MemHit(10^6)$	7.7	14.0	17.9	39.7	55.1	64.8	88	111	108.4	150.5	124	159.4	247.1	1400
$L_3Hit(10^6)$	1.7	3.1	3	6	6.6	8.0	10.2	13.5	13.9	19.7	15.4	24.7	43.0	345.5
$L_2Hit(10^6)$	7.0	14.9	21.2	42.5	65.1	78.7	107	136	136.5	193.9	159.5	171.8	227.8	801.3
$L_1Hit(10^6)$	942.3	2000	3300	6500	11400	12500	16600	19200	19100	12300	20500	8700	12500	61800
$RemoteAccessRatio$	0.4686	0.4548	0.4682	0.4470	0.4461	0.4437	0.4457	0.5750	0.4365	1.1559	0.4352	0.4356	0.6050	0.2099
$t_{hit2}(ns)$	24.7	24.2	24.7	23.9	23.8	23.7	23.8	29.0	23.5	52.2	23.4	23.4	30.2	14.4
$t_{hit3}(ns)$	38.7	38.2	38.7	37.9	37.8	37.7	37.8	43.0	37.5	66.2	37.4	37.4	44.2	28.4
$t_{mem}(ns)$	90.8	90.0	90.8	89.6	89.5	89.4	89.5	96.6	89.0	128.6	88.9	89.0	98.3	76.5
$T_{tot}(s)$	3.03	6.03	8.98	18.10	29.87	33.13	44.30	53.89	51.78	55.48	58.51	71.80	174.89	2386.50
$t_{app}(s)$	3	6	9	18	30	33	44	54	52	56	59	72	90	716
% of Memory Required	2.24	4.47	8.94	17.88	35.76	44.70	53.64	62.58	67.06	71.53	76.00	78.23	80.47	82.70
$error(\%)$	-0.85	-0.42	0.26	-0.54	0.42	-0.39	-0.67	0.21	0.42	0.93	0.83	0.28	-94.33	-233.31

Table 3.22: Experimental results with 2 GiB Memory Configuration($\hat{Hm_2}$): Nehalem micro-architecture

N(millions) / Metrics	1	2	4	8	16	32	56	60	62	64	68	70	72	74	76	77
$L_2HitRatio$	0.2505	0.2640	0.1548	0.2031	0.2009	0.1918	0.1850	0.1865	0.2265	0.2403	0.3163	0.3371	0.3592	0.3322	0.3136	0.1386
$L_3HitRatio$	0.2433	0.1101	0.1112	0.0868	0.0583	0.0422	0.0329	0.0311	0.0365	0.0321	0.0321	0.0278	0.0276	0.0383	0.0773	0.1724
$L_2Miss(10^6)$	2.1	7.3	17.1	13.2	23.7	44.3	77.3	115.7	215.1	101.5	160	171.6	317.2	1600	191.4	3100
$L_3Miss(10^6)$	1.7	6.5	16.4	12	21.7	40.5	72	109.7	206.8	94.7	155	163.6	307.4	1500	171.3	2500
$Mreads + Murites(10^9B)$	7.8	16.2	35.1	73.6	140.8	251.0	472.0	818.0	1600	608.3	4900	1100	2200	11600	737.5	1000
$QPI_{socket}(10^9B)$	0.0511	1.1	2.3	6.1	14.2	41.7	77.1	79.0	269.4	326.2	2200	277.9	684.2	1000	569.1	281.9
$L_1HitRatio$	0.9948	0.9887	0.9867	0.9955	0.9957	0.9955	0.9942	0.9916	0.9795	0.9930	0.9870	0.9860	0.9630	0.9630	0.9630	0.9630
$DiskHit$	395.6	395.5	395.3	396.1	394.8	392.6	384.2	382.4	380.6	378.7	376.7	374.4	373.1	2900	79700	360900
$MemHit(10^6)$	1.7	6.5	16.4	12	21.7	40.5	72	109.7	206.8	94.7	155	163.6	307.4	1500	171.3	2500
$L_3Hit(10^6)$	0.5513	0.8071	2	1.1	1.3	1.8	2.5	3.5	7.8	3.1	5.1	4.7	8.7	61	14.3	526.3
$L_2Hit(10^6)$	0.6986	2.6	3.1	3.4	6.0	10.5	17.6	26.5	63	32.1	74	87.2	177.8	776	87.4	498.3
$L_1Hit(10^6)$	533.6	871.5	1500	3700	6900	12100	16300	16800	13300	19000	17800	18200	12900	60800	7300	93600
$RemoteAccessRatio$	0.0065	0.0665	0.0650	0.0830	0.1008	0.1660	0.1633	0.0966	0.1645	0.5362	0.4432	0.2484	0.3066	0.0902	0.7717	0.2791
$t_{hit2}(ns)$	6.3	8.7	8.6	9.3	10	12.6	12.5	9.9	12.6	27.4	23.7	15.9	18.3	9.6	36.9	17.2
$t_{hit3}(ns)$	20.3	22.7	22.6	23.3	24	26	26.5	23.9	26.6	41.4	37.7	29.9	32.3	23.6	50.9	31.2
$t_{mem}(ns)$	65.4	68.7	68.6	69.6	70.5	74.1	74	70.3	74	94.5	89.4	78.7	81.9	70.0	107.4	80.3
$T_{tot}(s)$	1.99	3.02	4.99	9.01	16.13	28.22	38.92	42.40	43.65	48.64	52.09	51.60	55.21	243.39	196.30	1136.87
$t_{app}(s)$	2	3	5	9	16	28	39	42	44	49	52	52	55	58	99	458
% of Memory Required	1.12	2.24	4.47	8.94	17.88	35.76	62.58	67.06	69.29	71.53	76.00	78.23	80.47	82.70	84.94	86.05
$error(\%)$	0.70	-0.76	0.24	-0.10	-0.83	-0.79	0.21	-0.95	0.79	0.74	-0.17	0.76	-0.39	-319.64	-98.28	-148.23

Table 3.23: Experimental results with 4 GiB Memory Configuration($\hat{Hm_3}$): Nehalem micro-architecture

N(millions) / Metrics	1	2	4	8	16	32	56	60	62	64	68	70	72	74	76	77	78	79
$L_2HitRatio$	0.2645	0.2540	0.2213	0.2149	0.2143	0.2084	0.4752	0.4905	0.4974	0.5059	0.5160	0.5183	0.5254	0.5277	0.5333	0.5372	0.5377	0.5383
$L_3HitRatio$	0.2335	0.2150	0.1496	0.0956	0.0671	0.0415	0.0355	0.0307	0.0339	0.0298	0.0292	0.0315	0.0303	0.0319	0.0294	0.0301	0.0342	0.0292
$L_2Miss(10^6)$	2.7	3.1	7.8	12.4	23.2	43.8	79.9	85.8	93.8	91.9	132.1	107.2	108.6	193.2	111.5	199.9	148.2	171.5
$L_3Miss(10^6)$	2.3	2.7	7.0	11.4	21.2	40.5	65.3 73.6	80.1	87.1	84.7	123.3	98.5	100	182.2	103.1	190	134.5	162.8
$Mreads + Mwrites(10^9B)$	11.4	14.3	42.8	76.1	146.5	284.4	395.9	420.8	450.4	426.5	654.8	482.1	483.4	1000	491.7	1000	620.6	840.7
$QPI_{socket}(10^9B)$	0.1386	0.1185	0.7444	4.2	9.4	22.6	88.1	100.9	112.7	114.6	162.1	134.2	133.8	142.6	164.5	807.6	493	308.4
$L_1HitRatio$	0.9960	0.9954	0.9943	0.9939	0.9930	0.9927	0.9890	0.9887	0.9884	0.9881	0.9821	0.9875	0.9871	0.9737	0.9870	0.9520	0.9790	0.9760
$DiskHit$	4	3.9	3.9	3.9	3.9	3.8	3.5	3.5	3.4	3.4	3.3	3.2	3.2	3.1	3.1	3	2.9	2.9
$MemHit(10^6)$	2.3	2.7	7.0	11.4	21.2	40.5	65.3 73.6	80.1	87.1	84.7	123.3	98.5	100	182.2	103.1	190	134.5	162.8
$L_3Hit(10^6)$	0.6909	0.7408	1.2	1.2	1.5	1.8	2.7	2.5	3.1	2.6	3.7	3.2	3.1	6.0	3.1	5.9	4.8	4.9
$L_2Hit(10^6)$	0.9644	1.1	2.2	3.4	6.3	11.5	72.4	82.6	92.8	94.1	140.9	115.4	120.2	215.8	127.3	232	172.4	199.9
$L_1Hit(10^6)$	908	903.7	1700	2600	4200	7500	13700	14700	15900	15400	15000	17600	17500	15100	18100	8600	14900	15100
$RemoteAccessRatio$	0.0122	0.0083	0.0174	0.0549	0.0638	0.0796	0.2225	0.2397	0.2502	0.2688	0.2475	0.2784	0.2768	0.1381	0.3346	0.8054	0.7943	0.3669
$t_{hit2}(ns)$	6.5	6.3	6.7	8.2	8.6	9.2	14.9	15.6	16	16.8	15.9	17.1	17.1	11.5	19.4	38.2	37.8	20.7
$t_{hit3}(ns)$	20.5	20.3	20.7	22.2	22.6	23.2	28.9	29.6	30.0	30.8	29.9	31.1	31.1	25.5	33.4	52.2	51.8	34.7
$t_{mem}(ns)$	65.7	65.5	66	68	68.5	69.4	77.2	78.2	78.8	79.8	78.6	80.3	80.2	72.6	83.4	109.3	108.7	85.2
$T_{tot}(s)$	1.99	2.01	4.00	5.98	9.92	18.00	34.23	37.11	40.25	39.32	42.01	45.16	45.20	46.16	47.44	47.08	51.27	48.38
$t_{app}(s)$	2	2	4	6	10	18	34	37	40	39	42	45	45	46	47	47	51	48
% of Memory Required	0.56	1.12	2.24	4.47	8.94	16.76	27.94	31.29	33.53	34.65	35.76	38.00	39.12	40.23	41.35	42.47	43.03	43.59
$error(\%)$	0.34	-0.71	0.07	0.33	0.76	0.02	-0.68	-0.30	-0.62	-0.81	-0.01	-0.36	-0.44	-0.36	-0.93	-0.18	-0.53	-0.80

Table 3.24: Experimental results with 6 GiB Memory Configuration($\hat{Hm}_4$): Nehalem micro-architecture

N(millions) / Metrics	1	2	4	8	16	32	56	60	62	64	68	70	72	74	76	77	78	79
$L_2HitRatio$	0.4208	0.4803	0.4942	0.5587	0.5844	0.6017	0.6160	0.6148	0.6149	0.6165	0.6163	0.6220	0.6178	0.6216	0.6243	0.6276	0.6262	0.6305
$L_3HitRatio$	0.1749	0.1436	0.1055	0.0587	0.0482	0.0326	0.0293	0.0277	0.0292	0.0306	0.0278	0.0276	0.0315	0.0360	0.0319	0.0314	0.0318	0.0291
$L_2Miss(10^6)$	3	5.8	14.6	19	35.2	62.5	100.3	106.3	109.6	111.2	115.4	118.4	160	161.7	137.3	127.5	134	191
$L_3Miss(10^6)$	2.6	5.2	13.5	18.3	33	59.8	94	99.3	101.6	104	108.7	111.9	150	151.4	125.8	118.3	122.7	181.8
$Mreads + Mwrites(10^9B)$	8.8	19.1	41.6	73.5	139.5	247.8	411.2	418.4	431.5	432.8	449.5	448.4	2300	644	506.8	473.4	526.4	793.5
$QPI_{socket}(10^9B)$	4.5	9.1	20.2	36.6	69.1	120.9	194.5	198.5	203.3	202.3	737.3	341.4	260.8	221	305.7	661.5	243.6	3600
$L_1HitRatio$	0.9939	0.9906	0.9828	0.9850	0.9827	0.9808	0.9800	0.9789	0.9789	0.9785	0.9500	0.9735	0.9750	0.9682	0.9740	0.9510	0.9775	0.9775
$DiskHit$	4.8	4.8	4.8	4.8	4.8	4.5	4.1	4	4	3.9	3.8	3.7	3.7	3.6	3.5	3.5	3.4	3.3
$MemHit(10^6)$	2.6	5.2	13.5	18.3	33	59.8	94	99.3	101.6	104	108.7	111.9	150	151.4	125.8	118.3	122.7	181.8
$L_3Hit(10^6)$	0.5614	0.8727	1.6	1.1	1.7	2	2.8	2.8	3.1	3.3	3.1	3.2	4.9	5.7	4.1	3.8	4.0	5.4
$L_2Hit(10^6)$	2.1	5.3	14.3	24.1	49.5	94.4	160.9	169.7	175.0	178.8	185.4	194.8	258.6	265.7	228.1	214.9	224.4	326
$L_1Hit(10^6)$	831.7	1200	1700	2800	4800	8000	12800	12800	13200	13200	5700	11500	16300	13000	13700	6600	15600	16700
$RemoteAccessRatio$	0.5072	0.4767	0.4857	0.4985	0.4951	0.4878	0.4730	0.4743	0.4711	0.4675	1.6404	0.7616	0.1138	0.3432	0.6031	1.3973	0.4627	4.5402
$t_{hit2}(ns)$	26.3	25.1	25.4	25.9	25.8	25.5	24.9	25.0	24.8	24.7	71.6	36.5	10.6	19.7	30.1	61.9	24.5	187.6
$t_{hit3}(ns)$	40.3	39.1	39.4	39.9	39.8	39.5	38.9	39	38.8	38.7	85.6	50.5	24.6	33.7	44.1	75.9	38.5	201.6
$t_{mem}(ns)$	92.9	91.2	91.7	92.4	92.2	91.8	91.0	91.1	90.9	90.7	155.2	106.9	71.3	83.9	98.2	141.8	90.5	314.7
$T_{tot}(s)$	2	3	4	8	14	24.02	38.27	39.01	40.12	40.39	41.85	42.24	46.17	44.17	46.78	43.67	47.90	119.82
$t_{app}(s)$	2	3	5	8	14	24	38	39	40	40	42	42	46	44	47	44	48	47
% of Memory Required	0.75	2.10	4.20	8.40	16.79	33.58	58.77	62.96	65.06	67.16	71.36	73.46	75.56	77.65	79.75	80.80	81.85	82.90
$error(\%)$	0.09	0.15	0.38	-0.37	-0.12	-0.10	-0.72	-0.03	-0.31	-0.97	0.37	-0.58	-0.42	-0.38	0.46	0.75	0.22	-225.35

Chapter 4

Executable models for interventional X-Ray

4.1 Introduction

Designed for interventional cardiac, brain and vascular procedures, Allura X-Ray system provides angiography that is extremely reliable and displays images of very high clarity. Diagnosis and treatments based on angiography provided by this equipment plays a gigantic role in saving lives. This makes Allura X-Ray system an extremely sophisticated and important equipment for critical medical decisions. In order to ensure that the system meets the expectations of customers, a series of comprehensive tests are conducted at Philips Healthcare before the system is delivered to the customer. Though these tests unearth several failures before product delivery to the customer, a failure rate of 6 % is still observed. The failures encompass system software, application software and hardware failures. Unlike a PC used by a layman, a deviation in the performance of the PC used in Allura system can result in loss of critical information required to make important medical decisions. Therefore, identifying these failures and removing underperforming PCs from the system assembly has been a research area of prime importance. This chapter presents the investigation done in this area and a brief description of the executable models that can be used to predict failure of the PCs based on the hardware resource usage metrics of the chosen application on the system.

4.2 The Failures

In the context of discovering the reason behind the failure of the PCs in the factory during the execution of Allura application, the rationale is :

- Software Failure: The application software performs according to the requirements on most of the PCs. The probability of failures due to all software problems encountered in the system and application software is about 3 %.
- Hardware Failure: This kind of failure could lead to corrupt images, data loss in the network and complete breakdown of the system. Extensive set of tests conducted in the factory(night batch tests) indicate that parts of the test fail when PCs are tested for performance. The parts that fail are indicators of problems in the hardware. For example, disk and ethernet connection check tests. Hardware failure rate is reported to be 3%.

We are not concentrating on the software failures because this is not important to the production process. The production process ensures that the hardware components delivered to the customer are reliable. The R & D investigates the software bugs and failures. This narrows down the object of interest of this thesis to hardware failures.

The architectural instances of the PCs that fail and those of the PCs that perform well are exactly the same. This motivates the comparison of PCs through quantitative data obtained from performance measurements.

4.3 Test Suite Description

Allura X-Ray system components are assembled and tested using a comprehensive suite of tests designed specifically for testing the hardware components. The tests emulate the behaviour of the Allura application. They are conducted all night repetitively for several days to stress the PCs used in the system. Though each of these tests conducted exercise specific functionalities of the PCs, we are interested in an application that stresses the PC and makes it fail if it is found to have hardware problem. Verify Image Processor Test (VIPT) fits this criteria. However, this test cannot be shipped to the manufacturer because of the sophisticated environmental set-up required and the need to run the tests repetitively for several days. Nevertheless, it can be used as an application that we want to model by recording its resource usage behaviour. Therefore, we choose VIPT as the application (application and program are used interchangeably hereafter) we want to model.

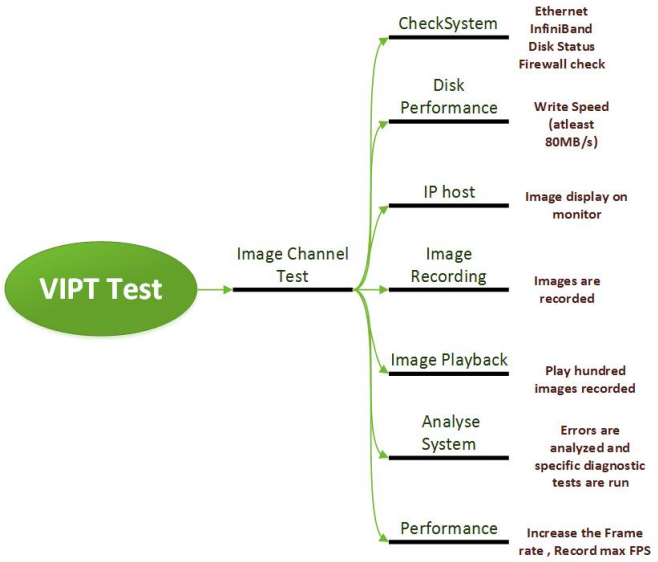


Figure 4.1: VIPT:Phase I

4.3.1 VIPT:

The purpose of VIPT is to test the Image Processing(IP) functionality of the IP subsystem (the IP software stack in combination with the IP hardware) without the hardware and software of other subsystems (e.g: Image Detection).

The test consists of a sequence of sub-tests. The result of each subset influences which subset will be run next, for example, if the test discovers disk problems, automatically disk tests are run.

The VIPT basically consists of two phases:

1. Phase I: In this phase, the system is stressed to use all its functionality. It also constantly listens to a fault-finding component in the system to detect abnormalities, if any. See Figure 4.1 for the tests conducted in Phase I. The tests use Field Service Component IP (FSCIP) which basically forms a user interface to the Field Service Engineer to start the VIPT. It also consists of a module, Performance Monitoring and Routing (PMR) that implements different functionalities required by the interfaces of the VIPT.

2. Phase II: In this phase, the VIPT runs tests needed to analyse the errors found (if any) during the first phase.

The series of tests, called Image Channel (IC) tests conducted during Phase I are as follows:

1. Init : This is used to start all required graphs and a special acquisition procedure that gets images from the disk. It shows monitor ID, PC Name, port details on all the monitors connected to the system and serves as a visual check of the monitor connections. These can be visualized in Figure 4.2. It then starts a test that listens to the system to find errors. If no errors are found, it continues with the sub-tests.

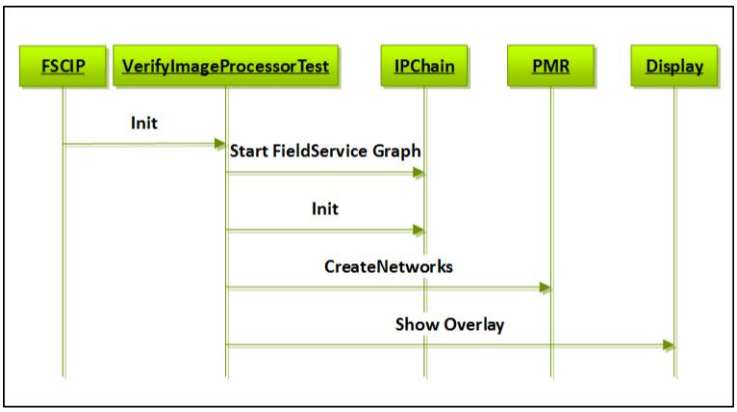


Figure 4.2: Init

2. Check System Test (IC1): This test checks a few hardware resources in the system which are easily diagnosable which are:

   - SMART data check for all disks
   - Quick ethernet check
   - Firewall check

   If problems are detected, diagnostic tests are scheduled to run during the second phase of the test.

3. Disk Performance Test(IC2): This step measures the write speed of the IP PC disks. The speed should be atleast 80 MB/s. The threshold of 80 MB/s is a trade-of between the expected write performance (which is specified as over 100 MB/s) and the minimally needed performance of at least 60 MB/s. The disk performance test tries to detect disk problems before they impact the functioning of the system.

4. Image Channel IP Host(IC3): In this step, 100 images are displayed on all the monitors(Reference, Exam, View) of the system. The test starts by setting up the correct networks and enabling full blown image processing on the used modules.The frame speed is set to 30 Frames per Second(FPS) and then 100 images are shown on the monitors. These steps are shown in Figure 4.3. When all images have been

shown, the Cyclic Redundancy Check [1](CRC) of all the images is checked against the expected value. This test is repeated for every monitor.

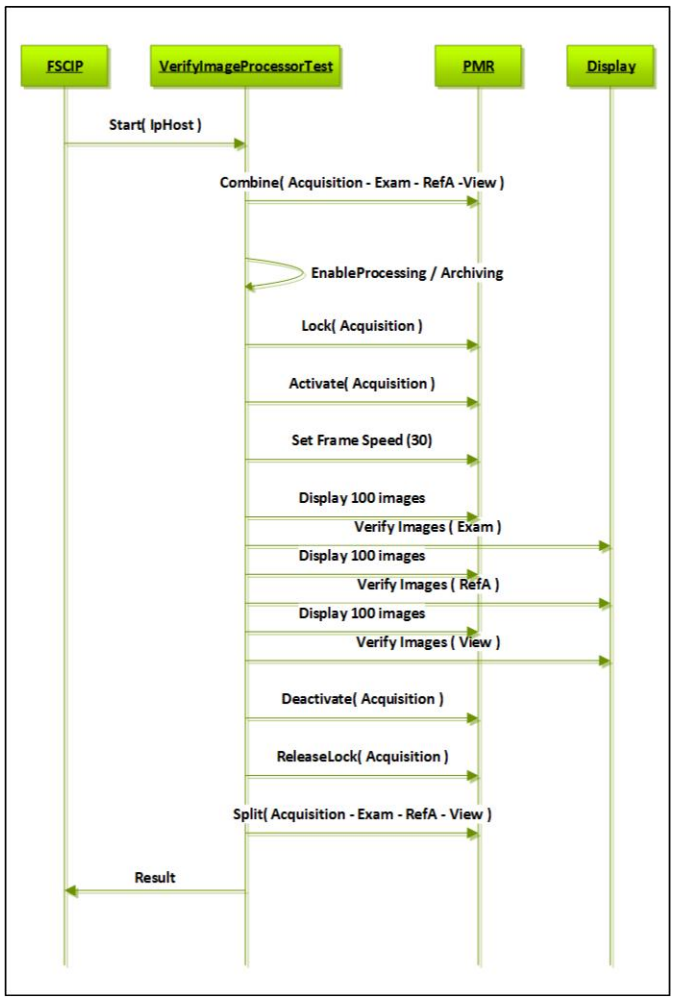


Figure 4.3: Image Channel IP Host

5. Image Channel Image Recording(IC4): In this step, a repository, Recorded Image List (RIL) is created and image processing is enabled. The frame speed is set to 30 FPS and then 100 images are recorded to the repository. If no errors occur, the test is successful. The steps can be visualised in Figure 4.4.

6. Image Channel Image Playback(IC5): The test starts by setting up the correct network (e.g, host to IP PC connection, Infini-Band cable connection, ethernet) using the PMR module and enables image processing. The repository which was recorded by the previous scenario is played on all the IP PC monitors(100 images).

[1]A cyclic redundancy check is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data

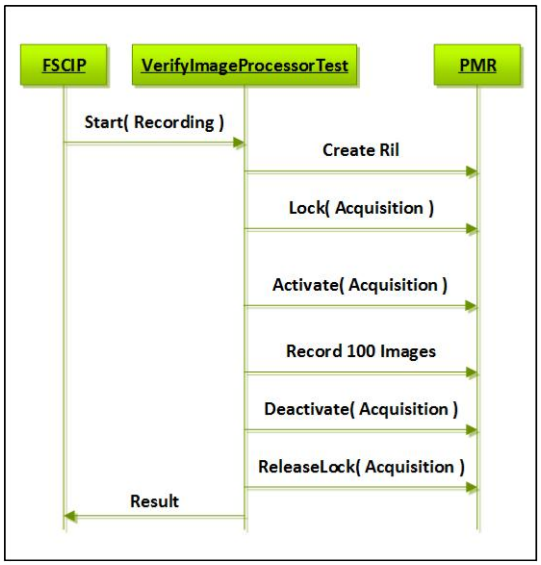


Figure 4.4: Image Recording

See Figure 4.5. The CRC of all images are checked against the expected values and if all the images are displayed correctly, the test is considered successful.

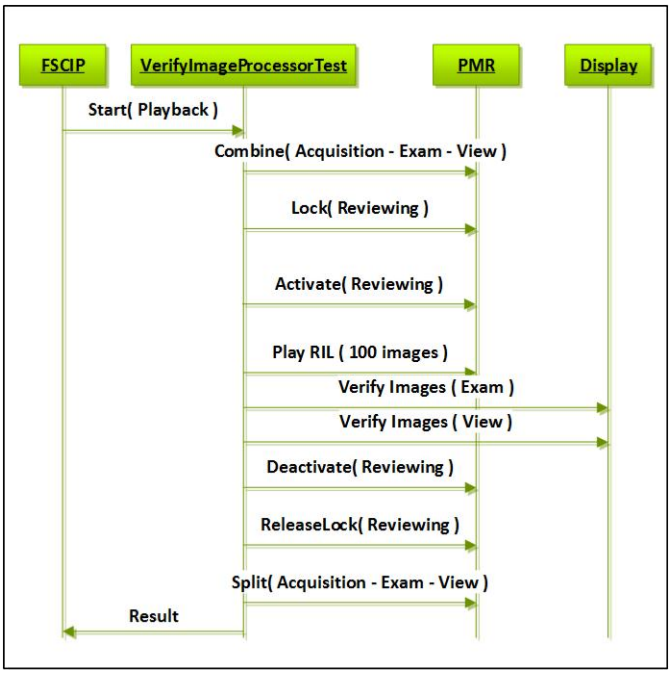


Figure 4.5: Image Playback

7. Analyse System (IC6):In this step all the errors from the system which have been

collected in the previous test steps are analysed and specific diagnostic tests are scheduled for suspicious hardware parts.

8. Image Channel Performance(IC7): If all previous steps have passed, the performance test is run. This test is not run when any error has been detected by the previous test, since we cannot be sure what we are testing in that case.

   Tests are conducted with a frame rate from 30-33 FPS. Once a frame rate of 33 FPS is reached, the test increases the frame rate (upto a maximum of 50 FPS) until the test fails. This is done to have an informal performance indicator of the system.

The series of tests run during Phase II are:

1. Disk Analysis Test: This is a self test run on the disks with problems. The tests are run on these disks in parallel.

2. Firewall Analysis Test: The firewall settings are checked on the host and the IP PC. It should be turned ON and a list of applications should be added to the exception list.

3. Infini-Band Analysis Test: When an Infini-Band(image link) problem has been detected during Phase I,this analysis test is run. It tries to determine the problem in the hardware by means of a loop-back connector which the service engineer has to place manually.

4. Ethernet Analysis Test: When an Ethernet problem has been detected during Phase I, this analysis test is scheduled to run. It tries to determine a problem with the Ethernet hardware by means of a field-service cable which the service engineer has to place manually.

Note that we have collected measurements only for Phase I of the VIPT.

4.4 Creation and Execution of Executable Models

As discussed in Chapter 1, the purpose of executable models is to predict the nature of the PC before production. The steps taken in this direction are listed below.

- Choosing the program $p1$: A program that can stress the PC and pave the way to finding the indicators of hardware problems had to be chosen. For the reasons presented in section 4.3 and the type of tests described in Section 4.3.1, VIPT is chosen as the program (application to be modelled), $p1$.

- Include required metrics in the tool: The MeasureLoadCLI is used to measure the RUM for the VIPT program using Intel PCM. It measures a rich set (more than hundred) of hardware metrics with finer details such as L_2, L_3 cache hits for individual cores along with the high level metrics such as system and socket metrics. However,we found that the QPI traffic was not included as one of the metrics. The tool is enhanced by including the incoming QPI traffic to the sockets. Note that Intel does not provide support for measuring the outgoing QPI traffic in Nehalem micro-architecture and any QPI traffic on Sandy Bridge micro-architecture.

- Include the MeasureLoadCLI tool in the factory tool-set: The tool was included in the factory tool-set using a script that concurrently runs the tool along with the VIPT. The metrics measured are logged into a text file for analysis.

The measurements are done for every one second. Some of the most important metrics measured are listed in Table 4.1.

Table 4.1: Hardware metrics from MeasureLoadCLI tool

core	socket	system
$m.Core_iUsage$	$m.Socket_jUsage$	$m.Usage$
$m.Core_iL_3Miss$	$m.Socket_jL_3Miss$	$m.L_3Miss$
$m.Core_iL_2Miss$	$m.Socket_jL_2Miss$	$m.L_2Miss$
$m.Core_iL_3HitRatio$	$m.Socket_jL_3HitRatio$	$m.L_3HitRatio$
$m.Core_iL_2HitRatio$	$m.Socket_jL_2HitRatio$	$m.L_2HitRatio$
-	$m.Socket_jM_r$	$m.M_r$
-	$m.Socket_jM_w$	$m.M_w$
-	$m.Socket_jQPI$	$m.QPI$

As seen from Table 4.1, the metrics are measured for core, socket and the system. The subscript $i = 1 \ldots 8$ for the core and $j = 1, 2$ for the socket. The definitions of the metrics are adopted from Chapter 3. The core, socket and system usage can be defined as:

$$Usage = \frac{Instructionsretired * Threadspercore}{Activecycles * maxIPC} * 100 \tag{4.1}$$

where,

- Instructions retired: The total number of instructions executed in the measurement interval

- Threads per core: This is an indication of whether Simultaneous Multi-Threading (SMT) is ON or OFF in the PC. If SMT is ON, the value of threads per core is 2. If SMT is OFF or not supported at all, the value of threads per core is 1. It is equal to 1 for the PCs used for our study.

- Maximum Instruction Per Cycle(maxIPC): The maximum number of instructions that can be executed per cycle. It is equal to 4 in Nehalem and Westmere micro-architectures.
- Active cycles: The total amount of clock ticks in the measurement interval.

4.4.1 Hardware model of the PC

Allura X-Ray system evolves along with the Intel roadmap of processor micro-architectures as discussed in Chapter3. The Allura system incorporates processors with newer micro-architectures. Not all the hardware models of these processors can be discussed here for brevity. For measurements and metrics, we use the hardware model representation used in Chapter 3. Refer to Figure3.2.

4.4.2 Decisions and Limitations

To design and execute executable models it was necessary to make valid assumptions and decisions. They are:

The Tools used : The MeasureLoadCLI tool is used to measure the hardware metrics and the RUMM tool is used to create and execute executable models. Note that the timers used in both the tools are different and this has to be taken into consideration if the measurements from both the tools are to be compared.

Filter Background Noise :It is important to filter the metrics measured from the background noise resulting from the tools used. For this purpose, measurements were done on an idle IP PC using MeasureloadCLI and RUMM tool. This has to be considered before deciding the threshold values of metrics used in creating executable models and the analysis that follows.

Hardware setting : The RUMM tool supports the creation and execution of executable models only on a dual socket PC. It was also required to turn the Non Uniform Memory Access(NUMA) on in the BIOS setting of the PC. In Section 4.6, there is a detailed explanation of the rationale behind this decision.

Data Availability : The factory environment setting for collecting the measurements from the PCs requires several rounds of testing to ensure it does not disrupt the system set-up. So, the data required for creating executable models is available only after the testing phase is completed. The executable models that can be used to predict hardware failures can be created and validated only when such PCs are made available.

4.4.3 RUMM Tool

As discussed in Chapter 2 , micro-benchmark codes are used to create executable models. The term micro-benchmark in this context refers to a piece of code which is designed to create a certain amount of load on a hardware resource. The amount of load is variable from zero (no load) to 100% (full load). The process of increasing or decreasing load on a hardware resource through a micro-benchmark is referred to as tuning.

The RUMM tool[4] created by Gebreweld uses micro-benchmark codes to create executable models. It is a dashboard tool that can be used for the following purposes:

1. Make measurements on a single/ dual socket PC
2. Analyse measurements using a Chart Tool
3. Create executable models for a two socket PC
4. Generate load on a PC by running the executable model created

We are interested in the functionalities of the RUMM tool listed in the steps 3 and 4.

4.5 Experiments and Results

In Chapter 2, we presented the activity diagram for creating and validating the executable models in Figures2.3,2.4,2.5. The first step is collecting the RUM of the program, $p1$ on any machine, m.

Quantitative analysis of data is a prerequisite to creating executable model,$x1$. This requires measurements from several PCs. This serves as a basis to classify the machine m into good/bad.

We collected measurements from 6 PCs denoted as $m1, \ldots m6$. Based on the results of several tests conducted on machines, $m1, m2, m3, m4$, they are classified as good PCs, which means that the machines show no signs of failure during night batch tests. They exhibit high performance. This is used as the ground to verify the models created. $m5$ and $m6$ are classified as bad PCs. $m5$ is a machine whose disk was found to be missing to the application software and $m6$ is a machine which showed failures during the night batch tests conducted. $m6$ is a machine which has hard-to-find hardware problem.

The profiling of $m1, \ldots m4$ is based on the measurements of $p1$ and is used as a reference to decide the threshold values of the metrics on any good PC. Although $m5$ is a bad PC, the reason of failure is known (missing disk). The ultimate aim of this part of

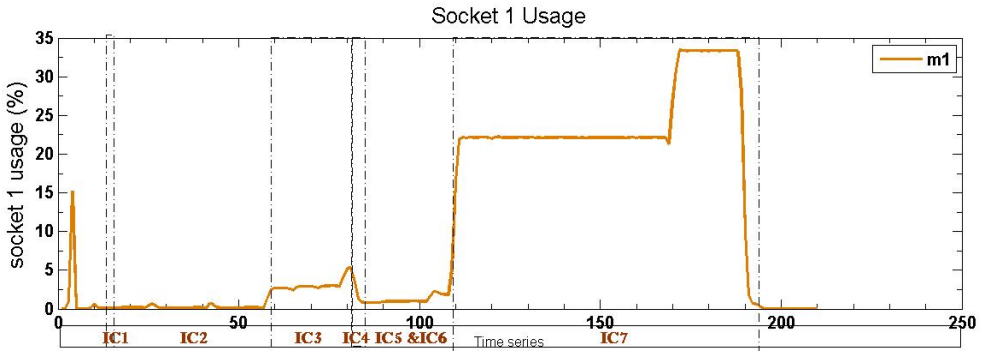
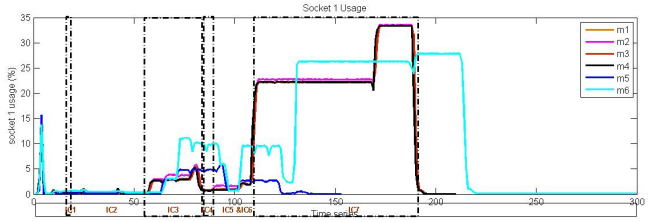


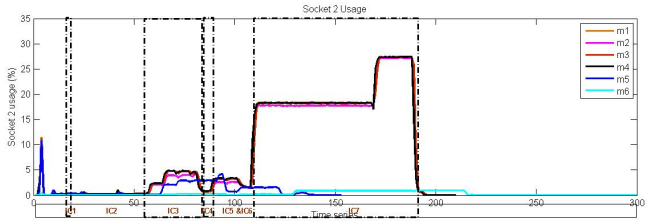
Figure 4.6: VIPT steps

thesis is to predict hard-to-find failures. $m5$ is still used in the study to compare the behaviour of two bad PCs, one with known reason ($m5$) and the other, with unknown reason ($m6$). The breakdown of the program, $p1$ chosen (VIPT) into steps described in the Section 4.3.1 is shown in Figure 4.6. Figure 4.6 shows the socket usage (%) for about 200 samples of measurement on $m1$.

The profiling of the machines $m1, \ldots m4$ is compared with machines, $m5$ and $m6$. $p1$ is run 25 times and 200 samples are collected, each time $m1, \ldots m5$ are tested and 300 samples of measurement are collected, each time $m6$ is tested. Figure 4.7, Figure 4.8 are plotted after taking the average of each sample from every trial for all the samples collected.



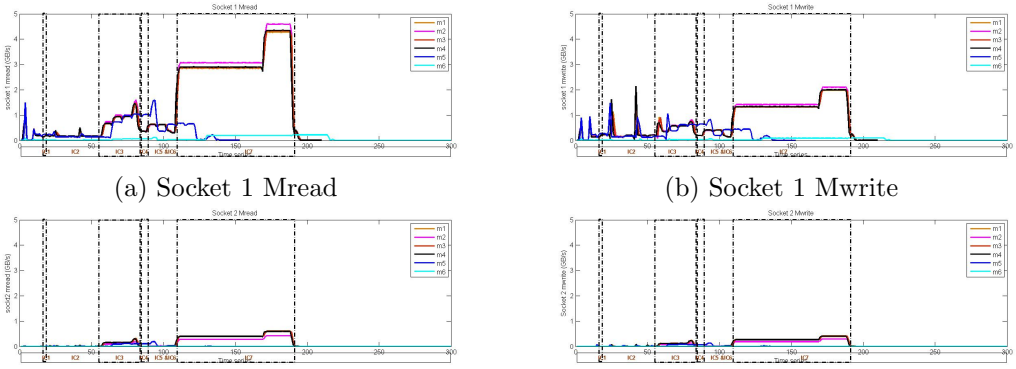
(a) Socket 1 Usage



(b) Socket 2 Usage

Figure 4.7: Hardware metrics: $\in K$. Usage(%) for both CPUs is plotted against the samples of measurement(time series). The steps of $p1$(VIPT) are also separately shown using dash-dotted lines.

The hardware metrics, socket(CPU) usages as shown in Figure 4.7 are denoted as $k1, k2 \in K$ for every machine selected. From Figure 4.7, it can be seen that $m5$ (in blue) and $m6$ (in cyan) exhibit change in the behaviour. They exhibit time shift in the values plotted along the x-axis and show significant changes in the behaviour during IC7. This is because, the performance of $m5$ is affected due to the missing disk;the system becomes slower and exhibits a time shifted pattern when compared to machines, $m1, \ldots m4$. $m5$ shows little or no performance metric numbers (predictable due to missing disk). $m6$, on the other hand, shows performance numbers during IC7, but are lower than the numbers of the set of good PCs. $m1, \ldots m4$ exhibit a similar pattern for all metrics observed along the x-axis. However, $m2$ (in pink) exhibits magnified performance numbers on socket 1 and reduced performance numbers on socket 2 when compared with machines, $m1, m3, m4$ along the y-axis. This suggests that machines which are classified as good PCs exhibit similar pattern, but admit small variations in the peak values along the y-axis. This has to be taken into account when deciding the threshold value of metrics when creating executable models. We can draw similar inferences from Figures 4.8.



(a) Socket 1 Mread (b) Socket 1 Mwrite

(c) Socket 2 Mread (d) Socket 2 Mwrite

Figure 4.8: Hardware metrics: $\in K$: The total amount of bytes read from(M_r) and written into (M_w) the memory (in GB/s) for both CPUs is plotted against the time series.They are represented as $k3, k4, k5, k6 \in K$ for every machine selected.

For all the hardware metrics plotted, we can see that the IC7 test distinguishes the behaviour of $m1, \ldots m4$ from $m5$ and $m6$. Therefore, $p1$ (VIPT) is a good indicator of hardware failures in a PC. Reflecting back to Figure 2.3, in Chapter 2, this can be verified using Equation 2.2.

The part of the thesis where the executable models are to be created and validated remains unfulfilled because the quantitative data from several PCs, good and bad was not available in time during the execution of the project. However, a suitable program that identifies difference in the pattern of the behaviour of PCs is identified and some metrics that exhibit difference in terms of time shift in performance numbers are demonstrated. It is also found that deciding a threshold value for metrics that are indicators

to failures need to take a range of values into consideration, considering the variations in performance numbers on good PCs. Quantitative data from many more PCs can be used to create reliable models that take the dynamics of behaviour of distinct PCs into consideration. Taking these factors into consideration, this work can be carried forward to achieve the intended goals. This is future work.

4.6 Lessons Learned

The experiments were conducted on a dual socket PC for reasons mentioned in the section 4.4.2. Since the PCs used had two physical sockets, we expected to see measurements for both the sockets. However, what we could see is that the PCM counters only displayed 1 physical socket and the measurements were also done for one socket only. After exploring the BIOS settings of the PCs used, it was seen that the NUMA feature was disabled for performance reasons known to Philips Healthcare.

When a separate DRAM is associated with every IMC and chip, platforms with more than one chip are NUMA. NUMA organisations have distinct performance advantages and disadvantages. As they are not a part of this project, they are not discussed in detail. But, it is important to understand that one can choose to populate all the memory banks by turning NUMA on or populate only one memory bank by tuning NUMA off. For applications that function only in the presence of two sockets, NUMA has to be turned on in the BIOS settings. Metrics such as QPI measurements can be measured only when NUMA is enabled. Therefore, we enabled NUMA.

Chapter 5

Conclusions and Future Work

In Cardiovascular minimal invasive interventions, physicians require low-latency X-ray imaging applications. This requires sufficient performance of the image-processing system while executing a plurality of functions. Multiple applications are executed in parallel and the PCs used for these systems have to offer sufficient performance under various use-cases. Adequate prediction and verification of the components used in the system based on modeling beforehand saves time and investment. They can also be given to hardware vendors for use in the selection process. In addition to this, models can be used to decide the optimum combination of applications on a specific hardware platform. This thesis proposes the creation of descriptive and executable models to be able to predict the performance of the system. More precisely, it consists of two different parts which are:

1. *To develop executable models using resource usage characteristics of an application run on a PC to be able to predict the nature of the PC (good/bad) before the production phase*

2. *To develop descriptive models to predict the execution of application(s),depending on the hardware model of the PC*

In the context of part I of the thesis, the aim is to create executable models that emulate the behaviour of the Allura application (or of any application that shows that the PC is faulty). These models unearth the causes of unexpected and hard-to-find failures in the PCs used in the Allura system. Executable models can be created using micro-benchmark codes. On running these models on a PC, from the measurements of the hardware metrics on that PC, the nature of the PC should be deduce-able. This kind of prediction offers significant cost reduction by avoiding shipping charges, that would otherwise happen due to the failure of the PCs, saves time by manufacturing PCs that meet the required specifications if the models are shipped to the manufacturer of the

PCs.

Creating executable models relies on the availability of measurements of various suspicious hardware metrics that are most likely to cause failures. A thorough analysis of the quantitative measurements from several PCs is definitely required to discover the agents of failure and create models that stress the hardware components that are to point to the agents of failure. However, for a sophisticated system like Allura X-ray system, collecting such measurements involves several steps such that they do not impair the functioning of the working system. The tools that are used to collect measurements need to be integrated to the already existing tool-set framework of the Allura System and this procedure requires rigorous and repetitive execution of tests to ensure reliability. This was one of the major challenges encountered during the execution of this project. The failure rate of the PCs used in the Allura system is less than 6 %. In order to be able to create executable models, it is required to be able to test and take measurements from PCs that are known to have hard-to-find hardware problems. This was one other major challenge encountered. We were able to test only one such PC. The Allura System executes a plurality of functions by the integration of several distinct PCs. The system functions with Allura application successfully only after careful integration of all the system components. Therefore, testing an IP PC with Allura application inevitably requires the successful set up of the entire cabinet of PCs used in the system. This task is complex and is another major challenge encountered in this project.

Due to the challenges discussed, it was not possible to complete the creation of executable models that predict the nature of the PC. However, some very useful and interesting observations were made that are useful for the future course of this part of the thesis. They are:

1. Effect of NUMA: The IP PCs used for study have two sockets (and so, two CPUs). The executable models use micro-benchmark codes that stress specific hardware components. The OS should be able to recognize both the sockets and their respective memory banks to be able to create and execute models. Also, metrics like QPI traffic are only captured for a two socket machine in Intel PCM. The OS distinguishes the two sockets physically only when the NUMA is enabled. For Allura X-ray systems, the NUMA feature is disabled because it is known to exhibit better performance with this feature tuned OFF using the BIOS settings of the PC. Therefore, to be able to create models and capture QPI metrics, it is necessary to enable NUMA.

2. Choice of the application: The creation of useful executable models that can actually predict failures depends on the correct choice of the application that, not only emulates the Allura application, but stresses the hardware components of interest specifically to observe unusual behaviour. After certain experiments and expert suggestions, it was found that VIPT, a part of the testing framework used for Allura systems can be used as the application (program) for which the measurements

can be collected.

3. Data Analysis: Data was successfully collected from four PCs with no hardware problems with the required settings. Several metrics such as the socket usages, socket cache misses, socket reads and writes from and to the memory were analysed to able to draw conclusions about the pattern of the behaviour of PCs with no problems. This step led to the inference that PCs with no hardware problems show high levels of performance during all the phases of VIPT (application chosen). They also demonstrate comparable level of numbers for hardware metrics that could be used to decide the threshold values of relevant metrics for all good PCs. The measurements collected from one PC with hard-to-find problems and one PC with missing disk measurements led to a definite observation that the PCs that are known to have hardware flaws exhibit drastic change in the performance numbers during IC7 of VIPT. They are also slower and show a time shift in the pattern when compared to the pattern of a set of good PCs. From the metrics observed it can also be seen that IC7 is the test that distinguishes the PCs based on their nature. Therefore, the data from this test only can be used for future analysis and observations.

These observations can be used to create executable models. Philips Healthcare has a toolset, RUMM that can be used to create executable models. However, these models can only create models for worst case resource usage scenario. This tool-set can be enhanced to be able to create executable models by considering all the measurement samples. The metrics such as QPI, cache hit ratios, cache misses are not a part of the modeling suite in RUMM. These fine level metrics, if included in the tool-set, can predict the nature of the PC along with the coarse level metrics already measurable and tunable in the tool-set. With these ideas and suggestions, this part of the thesis can be executed further to achieve the first goal of the thesis.

In the context of part 2 of the thesis, due to the challenges mentioned, it is not easy to incorporate the Allura application on an IP PC. However, as an initial step, descriptive models can be created using other featured applications such as STREAM. Descriptive models are created using the measurements collected on IP PCs using MeasureLoadCLI tool and Perfmon utility that capture several fine level metrics extremely useful for this part of the assignment. The contribution of this part of the thesis are:

1. Performance models that are used to predict the performance of the same PC with different memory configurations: The performance models created is used to predict the latency of an application on a PC with a certain hardware model. These models use the resource usage characteristics of the application on hardware platforms. Using this as the key principle, the resource usage characteristics of the same application on a slightly modified hardware model is predicted. We have chosen the same IP PC with different memory configurations as hardware models to predict the performance of STREAM. These models can not only predict the

performance, but also give performance numbers that can be used to compare different hardware configurations for an application. In the due course of this part of the thesis, other observations were also made, the most important among them being:

(a) Effect of Asymmetric Memory Distribution: It is seen and proved using the models created and measurements collected from tools and STREAM that the performance of a two socket PC with asymmetric memory configurations is found to be lower than the performance of the same PC with lesser memory capacity, but, distributed equally among the memory banks of the CPUs. Quantitatively, the performance degradation that happens due to asymmetric memory distribution is approximately equal to 20 %.

(b) Effect of the OS: It is also noted that the utilisation of the total memory installed and thus, the performance of the application on the PC used depends on the OS used. A 32 bit OS can only support a maximum of 2^{32} =4 GiB of memory. Therefore if a 32 bit OS is installed on a PC with 6 GiB memory configuration, cost reduction can be achieved by removing the unused memory (3 GiB) memory installed. Additionally, the performance of STREAM on the PC with 4 GiB of memory installed is also found to be better than the performance of the same PC with 6 GiB of memory installed because the OS does not recognise 3 GiB of memory installed on socket 2.

2. Performance models that are used to predict the performance of PCs with different micro-architectures: The performance models created are used to predict the latency of an application on a PC with entirely different hardware model. This can be put to use to evaluate the performance of the Allura system that evolves with the Intel processor roadmap. If the performance of the PC with new micro-architecture can be predicted, it can be used as a performance forecast before evolving into newer micro-architectures. This can be used to study the pros and cons of every micro-architecture before incorporating it as a part of the Allura system. During the execution of this part of the thesis, it was found that the PC with Sandy Bridge micro-architecture exhibits better performance than the PC with Nehalem micro-architecture. The models can also be used to compare the performances of PCs with different micro-architectures. The PC with the Sandy Bridge micro-architecture shows a performance improvement of 26 % over the best possible configuration of the PC with Nehalem micro-architecture(4 GiB memory configuration). This is also validated using the measurements from STREAM. The most interesting inferences made during the course of this part of the thesis are :

(a) Differences in the micro-architecture: It was seen that every aspect of the Sandy Bridge processor was redesigned by Intel to achieve better performance. The most important architectural enhancements were the inclusion of L_0 cache, L_3 cache on die, increased number of memory channels, increase in

the memory frequency, increase in the number of QPI links and its frequency, change in the snooping protocol and introduction of Turbo Boost technology. The performance improvements due to these architectural changes are predicted and incorporated in the models. The models created are validated on the PC with Sandy Bridge micro-architecture.

(b) Effect Of Turbo Boost: It was seen that the processor frequency ramps up to provide increased level of performance and then tunes down to create thermal reserves in the PC with Sandy Bridge micro-architecture. This affects the performance of the application.

The descriptive models created for both objectives can be used to predict the performance of the PCs for real time applications such as Allura. The set of descriptive models can be extended and enriched to study other performance properties such as throughput, bandwidth for network, graphic cards as well. This broadens the scope of comparison and results in better performance evaluation of systems with higher workloads and real time constraints. The enriched set of descriptive models created can be used to achieve significant cost reduction by combining two or more PCs into one PC for a multi-functional system with several PCs such as the Allura system.

References

[1] R. Albers, "Modeling and control of image processing for interventional X-Ray," Ph.D. thesis, Eindhoven University of Technology, 2010.

[2] "Roentgen's discovery of the X-Ray." [Online]. Available: http://www.bl.uk/learning/artimages/bodies/xray/roentgen.html

[3] "Interventional X-Ray." [Online]. Available: http://www.healthcare.philips.com/in_en/products/interventional_xray/Product/

[4] A. T. Gebreweld, "Resource-usage Modeling Tools for Performance Control," PDEng. thesis, Eindhoven University of Technology, September 2012.

[5] A. C. J. Kienhuis, "Design Space Exploration of Stream-based Dataflow Architectures," Ph.D. thesis, Delft University of Technology, January 1999.

[6] H. Koziolek et al, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, vol. 67, no. 8, pp. 397–408, August 2010.

[7] S. Becker, H. Koziolek, and R. Reussner, "Model-based Performance Prediction with the Palladio Component Model," in *Proceedings of the 6th international workshop on Software and performance*, 2007, pp. 54–65.

[8] "Prime95," June 2008. [Online]. Available: http://www.playtool.com/pages/prime95/prime95.html

[9] "STREAM." [Online]. Available: http://www.cs.virginia.edu/stream/ref.html

[10] J. D. McCalpin, "Sustainable Memory Bandwidth in Current High Performance Computers," October 1995. [Online]. Available: http://www.cs.virginia.edu/~mccalpin/papers/bandwidth/bandwidth.html

[11] "Perfmon Sample: Demonstrates How to Monitor System Performance Using Performance Counters." [Online]. Available: http://msdn.microsoft.com/en-us/

library/aa645516(v=vs.71).aspx

[12] T. Rolf, "Cache Organization and Memory Management of the Intel Nehalem Computer Architecture," December 2009.

[13] A. Semin, "Inside Intel Nehalem Microarchitecture," Trondheim, Norway, May 2009.

[14] M. E. Thomadakis, "The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms," March 2011.

[15] D. Levinthal, *Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors*, Intel Corporation.

[16] "Performance Monitor Counters." [Online]. Available: http://technet.microsoft.com/en-us/library/cc768048.aspx

[17] "Intel Tick Tock Model." [Online]. Available: http://www.intel.com/content/www/us/en/silicon-innovations/intel-tick-tock-model-general.html

[18] "HP Z220 SFF, Z220 CMT, Z420, Z620, and Z820 Workstations Maintenance and Service Guide." [Online]. Available: http://bizsupport2.austin.hp.com/bc/docs/support/SupportManual/c03678225/c03678225.pdf

[19] "Intel Xeon Processor E5-2650." [Online]. Available: http://ark.intel.com/products/64590/

[20] D. Kanter, "Intel's Sandy Bridge Microarchitecture," September 2010. [Online]. Available: http://www.realworldtech.com/sandy-bridge/

[21] ——, "Intel's Quick Path Evolved," July 2011. [Online]. Available: http://www.realworldtech.com/qpi-evolved/

[22] "Sandy Bridge," July 2013. [Online]. Available: http://en.wikipedia.org/wiki/Sandy_Bridge

[23] "Memory Performance of Xeon E5-2600/4600 (Sandy Bridge-EP) based Systems," Fujitsu Technology Solutions, Tech. Rep., July 2012.

[24] "Nehalem (microarchitecture)," July 2013. [Online]. Available: http://en.wikipedia.org/wiki/Nehalem_(microarchitecture)

[25] T. P. Morgan, "Intel plugs both your sockets with Jaketown Xeon E5-2600s," March 2012. [Online]. Available: http://www.theregister.co.uk/2012/03/06/intel_xeon_2600_server_chip_launch/

[26] J. Chang, "Intel Server Strategy Shift with Sandy Bridge EN and EP," May 2012. [Online]. Available: http://sqlblog.com/blogs/joe_chang/archive/2011/11/29/intel-server-strategy-shift-with-sandy-bridge-en-ep.aspx

[27] D. Kanter, "Sandy Bridge-EP Launches," March 2012. [Online]. Available: http://www.realworldtech.com/sandy-bridge-ep/

[28] A. L. Shimpi and G. Key, "The Dark Knight:Intel's Core i7," November 2008. [Online]. Available: http://www.anandtech.com/show/2658/4

[29] D. Kanter, "The Common System Interface: Intel's Future Interconnect," August 2007. [Online]. Available: http://www.realworldtech.com/common-system-interface/6/

[30] A. L. Shimpi, "Intel's Sandy Bridge Architecture Exposed," September 2010. [Online]. Available: http://www.anandtech.com/show/3922/intels-sandy-bridge-architecture-exposed/8

[31] M. Schuette, "Intel's Sandy Bridge Architecture & CPU Performance," January 2011. [Online]. Available: http://www.lostcircuits.com/mambo//index.php?option=com_content&task=view&id=98&Itemid=1

[32] D. Kanter, "Sandy Bridge for Servers," July 2011. [Online]. Available: http://www.realworldtech.com/sandy-bridge-servers/

[33] J. D. Gelas, "The Xeon E5-2600:Dual Sandy Bridge for Servers," March 2012. [Online]. Available: http://www.anandtech.com/show/5553/the-xeon-e52600-dual-sandybridge-for-servers

[34] "Intel 64 and IA-32 Architectures Software Developer's Manual," January 2013. [Online]. Available: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

[35] S. Jarp et al, "Evaluation of the Intel Sandy Bridge-Ep server processor," CERN openlab, Tech. Rep., March 2012.

[36] G. Kocchar et al, "Optimal BIOS settings for HPC with Dell Poweredge 12th generation servers," DELL, Tech. Rep., July 2012.

[37] "core-i7-processor-family-vtune-sw-opt-guide-1.1," December 2010. [Online]. Available: http://software.intel.com/en-us/articles/using-intel-vtune-performance-analyzer-to-optimize-software-for-the-intelr-coretm-i7-processor-family