POLITECNICO DI MILANO

FACOLTA DI INGEGNERIA DELL'INFORMAZIONE

MASTER OF SCIENCE IN ENGINEERING OF COMPUTING SYSTEMS

BENCHMARKING AND MODELING OF MULTI-THREADED MULTI-CORE PROCESSORS

Supervisor:	Prof. Paolo CREMONESI
Co-Supervisor:	Mr. Andrea SANSOTTERA

Master Thesis of

Rustam SAIDOV, MATRICOLA: 764648

ACCADEMIC YEAR 2012/2013

Abstract

In this master thesis a specific type of multithreading technique – Simultaneous Multithreading (SMT) is investigated. All experiments are conducted with Intel's proprietary implementation of it, officially referred to as Intel Hyper - Threading Technology (HTT), which in fact represents two-way SMT.

Simultaneous Multithreading aims to improve the utilization of processor resources by exploiting thread-level parallelism at the core level, resulting in an increase in overall system throughput. Intel uses this technology ubiquitously in its latest series of processors.

An extensive number of benchmark runs have been performed to understand the performance impact of HTT resulting in around 90 hours of run under different configurations of the system under test (SUT). Two different benchmarking suites with their own features and measuring approaches have been utilized for this purpose: SPEC Power SSJ 2008 benchmark and a synthetic benchmark stressing both ALUs and memory hierarchy. It has been experimentally proven that SMT, and HTT in particular, has the potential to dramatically improve utilization of the processor, increase overall throughput of the system (up to 33%) and decrease the system response time.

Unfortunately, modeling of SMT in Queuing Networks is still an open problem. Hence, we propose two Queuing Network models able to adequately predict performance impacts enabled by the technology. The first model is based on a birth-death Markov chain and the second is based on a Queuing Network with Finite Capacity Region (FCR). We validated the two proposed models on the datasets obtained from benchmark runs and observed that they achieve good accuracy with estimation error within the 3% - 10% interval. Lastly, we performed extensive comparisons between our models and the state of the art.

Sommario

In questa tesi prendiamo in considerazione una particolare tecnica di multithreading, il Simultaneous Multithreading (SMT). Gli esperimenti sono stati effettuati sull'implementazione proprietaria realizzata da Intel, chiamata ufficialmente Intel Hyper-Threading Technonlogy (HTT), che è sostanzialmente un SMT a due vie.

Il Simultaneous Multithreading ha lo scopo di migliorare l'utilizzo delle risorse del processore sfruttando il parallelismo a livello di thread, risultando quindi in un aumento del throughput del sistema. Intel usa questa tecnologia nelle più recenti serie dei sui processori.

Sono stati effettuati un vasto numero di benchmark per valutare l'impatto dell'HTT, per un totale di 90 ore, utilizzando differenti configurazioni del sistema di test (System Under Test - SUT). Due differenti suite di benchmark, con differenti caratteristiche e approcci di misurazione, sono state utilizzate per questo scopo: SPEC POWER SSJ 2008 e un benchmark sintetico che stressa sia le ALU che la gerarchia di memoria. Abbiamo mostrato sperimentale che il SMT, e l'HTT in particolare, hanno il potenziale di migliorare l'utilizzo del processore, incrementare drammaticamente il throughput del sistema (sino al 33%) e abbassare il tempo di risposta del sistema.

Sfortunatamente, modellare il SMT nelle reti di code è ancora un problema aperto. Pertanto, proponiamo due modelli a code in grado di predire l'impatto prestazione della tecnologia e valutiamo la loro accuratezza sui data-set ottenuti dall'esecuzione dei benchmark. Il primo modello è modello è basato una catena di Markov nascita-e-morrte e il secondo è basato sulle reti di code con regioni a capacità finita (Finite Capacity Region - FCR). Abbiamo validato i due modelli posti sui dataset ottenuti dall'esecuzione dei benchmark e abbiamo osservato che ottengono una buona precisione con un errore di stima nell'intervallo 3%-10%. Infine, abbiamo effettuato un confronto esaustivo con lo stato dell'arte.

Acknowledgments

To my family and friends who have helped and supported me in this experience.

To my co-supervisor Andrea Sansottera for his immense support,

encouragement throughout the thesis and

Prof. Paolo Cremonesi for his valuable time, comments and ideas.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Continuous progress in manufacturing technologies results in the performance of microprocessors that has been steadily improving over the last decades, doubling every eighteen months (Moore's law [26]). At the same time, the capacity of memory chips has also been doubling every eighteen months, but the performance has been improving less than ten percent per year [21]. The latency gap between the processor and its memory doubles approximately every six years, and an increasing part of the processor's time is spent on waiting for the completion of memory operations. Matching the performances of the processor and the memory is an increasingly difficult task [10][21]. In effect, it is often the case that up to sixty percent of execution cycles are spent waiting for the completion of memory accesses.

Many techniques have been developed to tolerate long-latency memory accesses. One such technique is instruction-level multithreading which tolerates long-latency memory accesses by switching to another thread (if it is available for execution) rather than waiting for the completion of the long–latency operation. If different threads are associated with different sets of processor registers, switching from one thread to another (called "context switching") can be done very efficiently.

In this work we analyze Simultaneous Multithreading (SMT) – the most advanced type of multithreading, where several threads can issue instructions at the same time. If a processor contains more than one pipeline, or it contains several functional units, the instructions can be issued simultaneously; if there is only one pipeline, only one instruction can be issued in each processor cycle, but the (simultaneous) threads complement each other in the sense that whenever one thread cannot issue an instruction (because of pipeline stalls or context switching), an instruction is issued from another thread, eliminating 'empty' instruction slots and increasing the overall performance of the processor. The objective of Simultaneous Multithreading is to substantially increase processor utilization in the face of both long instruction latencies and limited available parallelism per thread. The biggest advantage of Simultaneous Multithreading technique is that it requires only some extra hardware instead of replicating the entire core.

Some time ago Intel announced about its proprietary implementation of SMT, later called Hyper – Threading Technology (HTT), which basically allowed each processor core physically present in the system to be addressed by the operating system as two logical or virtual cores, and to share the workload between them when possible.

1.2 Motivation

The multithreading paradigm has become more popular as efforts to further exploit instruction level parallelism. This allowed the concept of throughput computing to reemerge to prominence from the more specialized field of transaction processing. Among different types of multithreading existing by far, Simultaneous Multithreading – the most advanced in terms of efficiency type of multithreading, became a common design choice due to its relative simple implementation and efficiency. As proof, the world largest and highest valued semiconductor chip maker - Intel Corporation, uses this technology ubiquitously in its latest series of processors. This fact motivated us to perform a series of experimental validations to understand real performance insights of the technology and the best usage practices of it, especially with regards to server-side equipment.

Even though the technology exist on the market for some years, yet there are only few adequate models have been proposed able to accurately predict performance gain enabled by the technology. This issue becomes of particular interest on a data centers scale, where it is extremely important to have precise estimations of data center capacity.

In fact, the presence of SMT significantly affects performance of the system, sometimes increasing it by one third over total capacity of the system, and in such a case utilization law, which establishes the linear relationship between the utilization and throughput of a computing systems and which is often intended as the basis for capacity planning in large data center, does not hold anymore. It can be clearly observed from the data we obtained for SPEC Power benchmark on system with 4 cores at 2660 MHz:

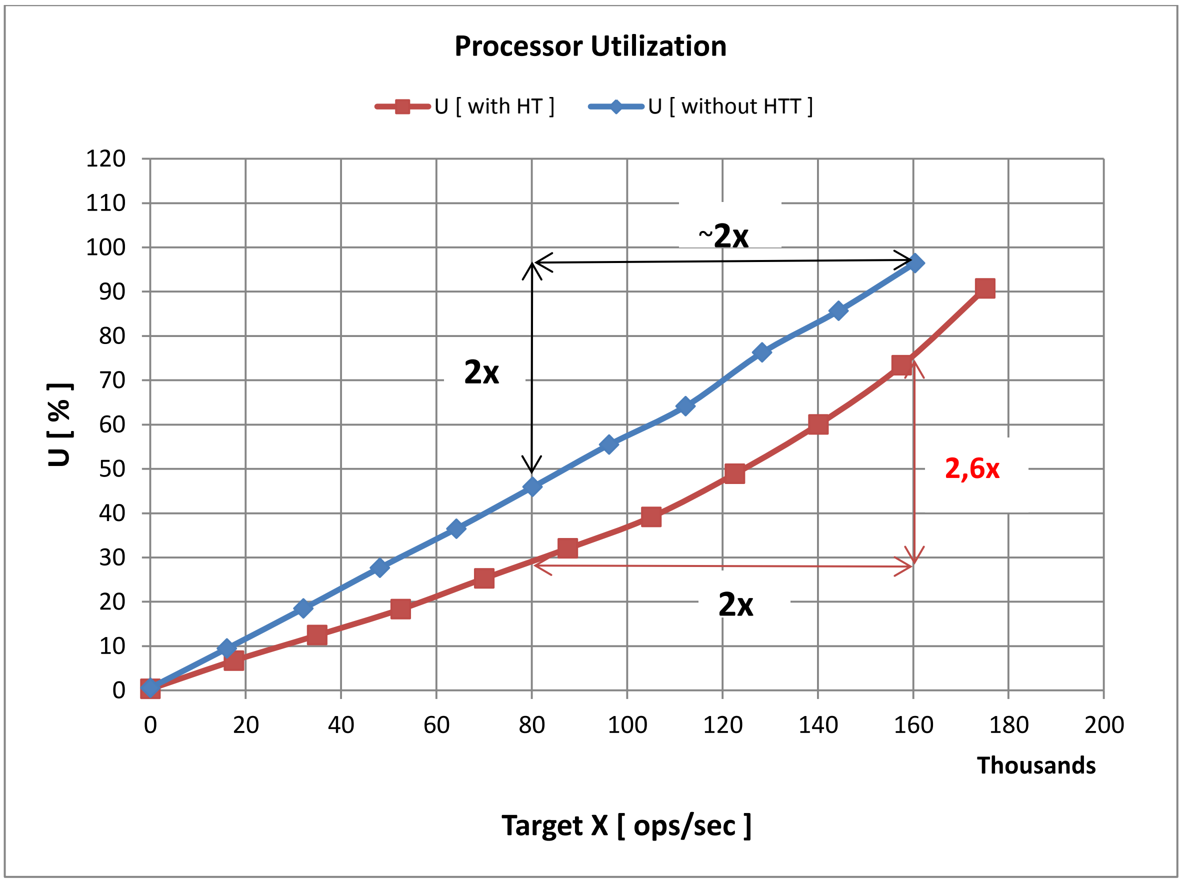

Figure 1.1: SPEC Power benchmark run: Processor
Utilization with HT {on, off} for 4cores at 2660 MHz

According to Utilization law, utilization must grow linearly with workload. As we can see from the Figure 1.1 this relation clearly does not hold, since utilization has increased 2.6 times while throughput just 2 times for the plot U [with HT].

Apparently, in such a case, the planed datacenter capacities would be heavily overestimated. This became one of the key moments led us to carry out research and to propose models for SMT/HTT processors possessing high predictive capacity and low estimation error.

1.3 Contributions

Thesis contributions are twofold. First, we perform a series of benchmark runs for different system configurations using two benchmarking suites. Second, we propose two different models to predict the performance (i.e. the response times and utilizations) of multi-core processors with SMT.

First model is an analytical birth-death Markov chain model based on two parameters and the second is Queuing Network model with Finite Capacity region. To estimate parameters of the first model three different techniques have been employed. Validation of the second model is performed using JMT (Java Modeling Tool)[23]. Lastly, the models are compared with other mentioned in the work state of art models.

1.4 Thesis outline

While this chapter aims at providing an overview of the work, reveals motivation, scope of the research work and performs brief introduction to the work, the remainder of the thesis is structured as follows:

- Chapter 2 provides some necessary background information about Simultaneous-Multithreading, Hyper-Threading technologies, as well as basic concepts about Queuing Network Models necessary for successful comprehension of the remainder.
- Chapter 3 presents two state of art QN models aimed at modeling HTT and SMT technologies.
- Chapter 4 presents results of benchmarking for different system configurations and reveals true potential of HTT.
- Chapter 5 introduces to two developed SMT/HTT models based on birth-death Markov chain model and Queuing Network model with Finite Capacity Region (FCR), as well as method intended to reduce model estimation error.
- Chapter 6 presents validation results for the proposed and mentioned in the work SMT/HTT models and models comparison results.
- Chapter 7 concludes by summarizing the contributions and discusses possible directions for the future research based on available results.

Chapter 2

Background

2.1 Introduction

Modern superscalar processors can detect and exploit high levels of parallelism in the instruction stream, being able to begin execution of high number of instructions every cycle. They do so through a combination of multiple independent functional units and sophisticated logic to detect instructions that can be issued (sent to the functional units) in parallel. However, their ability to fully use this hardware is ultimately constrained by instruction dependencies and long-latency instructions limiting available parallelism in the single executing thread. The effects of these, as an example, are shown as vertical waste (completely wasted cycles) and horizontal waste (wasted issue opportunities in a partially-used cycle) in Figure 2.1, which depicts the utilization of issue slots on a superscalar processor and capable of issuing four instructions per cycle.

Multithreaded architectures employ multiple threads with fast context switching between threads. Fast context switches require that the state (program counter and registers) of multiple threads be stored in hardware. Latencies in a single thread can be hidden with instructions from other threads, taking advantage of inter-thread parallelism (the fact that instructions from different threads are nearly always independent).

Simultaneous Multithreading (SMT) and Intel's implementation of it – Hyper-Threading (HTT), combines the superscalar's ability to exploit high levels of instruction-level parallelism with multithreading's ability to expose inter-thread parallelism to the processor. By allowing the processor to issue instructions from multiple threads in the same cycle, simultaneous multithreading uses multithreading not only to hide latencies, but also to increase issue parallelism. SMT and HTT can deal with both vertical and horizontal waste, as shown in Figure 2.2 and Figure 2.3.

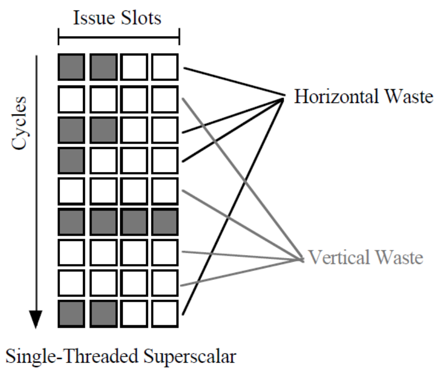

Figure 2.1: Utilization of issue slots on a superscalar processor.

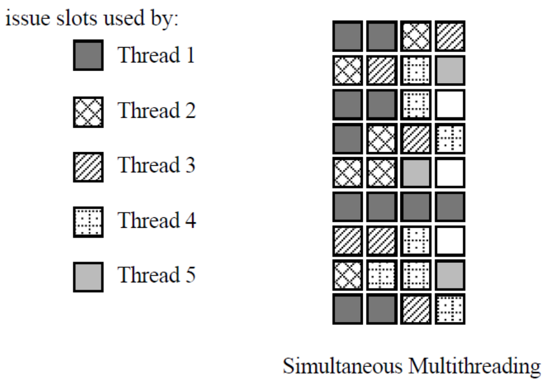

Figure 2.2: Utilization of issue slots on a superscalar SMT processor with 5 threads.

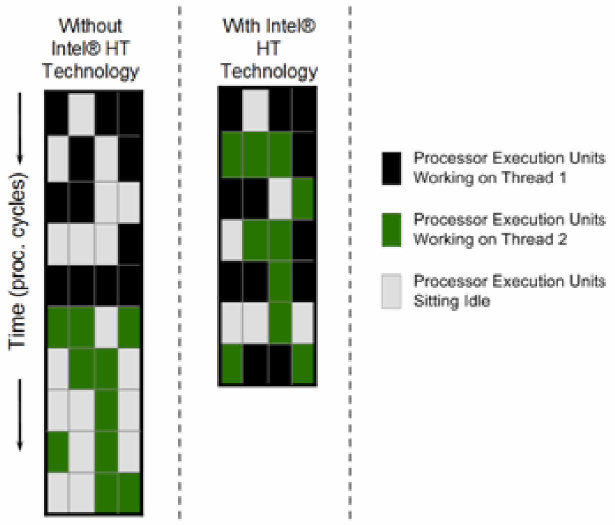
Without Intel® HT Technology
With Intel® HT Technology
Time (proc. cycles)
Processor Execution Units Working on Thread 1
Processor Execution Units Working on Thread 2
Processor Execution Units Sitting Idle

Figure 2.3: Utilization of issue slots on Intel's Nehalem based processor with HTT.

In Intel’s HTT the key hardware mechanism underlying this capability is an extra *architectural state* supported by the hardware [1]. Each thread in the processor is simply an instruction stream associated with a particular hardware context, as shown in Figure 2.4. Thereby each architectural state can support its own thread.

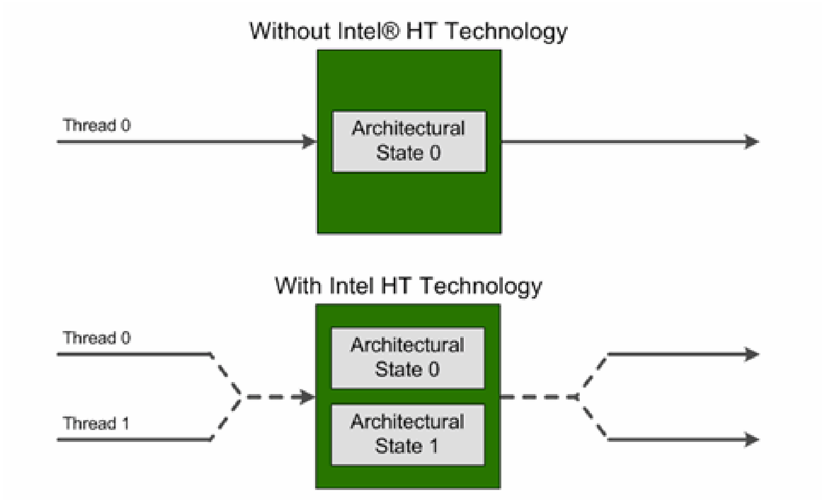

Figure 2.4: Intel’s HT Technology enables a single processor core to maintain two architectural states.

Many of the internal microarchitectural hardware resources are shared between the two threads. Operating systems and applications can schedule processes or threads on those logical processors. When one thread is waiting for an instruction to complete, the core can execute instructions from another thread without stalling. In addition, two or more processes can use the same resources. If one process fails then the resources can be readily re-allocated. The performance impact varies, depending on the nature of the applications running on the processors and on how the hardware is configured, but in most of the cases, Intel’s HTT and SMT allow substantial increase of the processor throughput, improving overall performance on threaded software.

2.2 The complexity of SMT

SMT processor is more complex than a corresponding single-threaded processor [8]. Potential sources of that complexity include:

- Fetching from multiple program counters — following multiple instruction streams will either require parallel access to the instruction cache(s) and multiple decode pipelines, or a restricted fetch mechanism that has the potential to be a bottleneck.
- Multiple register sets — accessing multiple register sets each cycle requires either a large single shared register file or multiple register files with a complex interconnection network between the registers and the functional units. In either case, access to the registers will be slower than in the single-threaded case.
- Instruction scheduling — in considered SMT processors, the scheduling and mixing of instructions from various threads is all done in hardware. In addition, if instructions from different threads are buffered separately for each hardware context, the scheduling mechanism is distributed (and more complex), and the movement of instructions from buffers to functional units requires another complex interconnection network.

2.2.1 Complexity of Intel's HTT

Intel processors can have varying numbers of cores, each of which can support two threads when Intel HT Technology is enabled [1]. For each thread, the processor maintains a separate, complete architectural state that includes its own set of registers as defined by the Intel 64 architecture [3] [5]. Some internal microarchitectural structures are shared between threads. Processors support SMT by replicating, partitioning or sharing existing functional units in the core. Hyper-threaded processor installed in our system under test has the following implementation policy [3][5]:

1. Replication — the unit is replicated for each thread.
   - register state
   - renamed RSB
   - large page ITLB
2. Partitioning — the unit is statically allocated between the two threads.
   - load buffer
   - store buffer
   - reorder buffer
   - small page ITLB
3. Competitive Sharing — the unit is dynamically allocated between the two threads.
   - reservation station
   - cache memories
   - data TLB
   - 2nd level TLB
4. SMT Insensitive — all execution units are SMT transparent

2.3 Introduction to Queuing Networks

Since substantial part of the research work is dedicated to construction of appropriate SMT/HTT models, in this section some background information about Queuing Networks is provided to reader, which will help him to better comprehend the rest of the work.

2.3.1 Queuing Network

During many years, computer and communication systems have been studied as a network of queues [16]. According to [17] Queuing network modeling is a particular approach to computer system modeling in which the computer system is represented as a network of queues which is evaluated analytically. A network of queues is a collection of service centers, which represent system resources, and customers, which represent users or transactions.

Figure 2.5 illustrates a single service center. Customers or transactions arrive at the service center, wait in the queue if necessary, receive service from the server, and depart. In fact, this service center and its arriving jobs constitute a queuing network model.

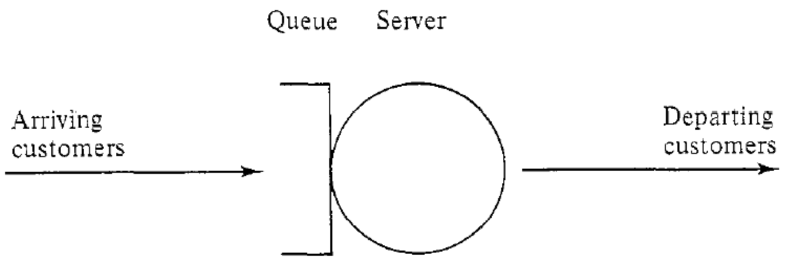

Figure 2.5: Single service center.

Here there are some definitions of queuing networks used in the work:

- **Class**: A class is a group of customers with each customer of that group having the same workload intensities (A_c, N_c, or N and Z_c) and service demand ($S_{c,d}$). A class MUST define the service demand ($S_{c,d}$) at each service center [17] [24].
- **Throughput**: Throughput is the number of customers received service at service center in a unit of time. Usually it has notation X_k, where *k* is a service center [17] [24].
- **Response Time**: Response Time is the average time a customer spends at a service center and has notation R_k [17] [24].
- **System Throughput**: System Throughput is the number of customers/transactions serviced by the system within a given period of time and has notation $X = \sum_k X_k$ [17] [24].
- **Visits**: Average number of visits at service center. Usually it has notation V_k, where *k* is a service center.
- **System Response Time:** System Response Time is the sum time a customer spends at

each service center. $R = \sum_k R_k * V_k$, where V_k is the average number of visits at station k [17][24].

- **Service center Utilization:** Service center Utilization is a ratio between the busy time (the time a service center is being used) and the whole measured period. This ratio describes the extent of utilization of a service center. Utilization could be greater than 0, but should be less than 1. Therefore, 0.5 utilization means half of the device's possible capacity is being used [17] [24].
- **Arrival Rate**: Arrival Rate is the rate of customers/transactions at which they arrive arriving to a queue in a unit of time and has notation λ_i [17] [24].
- **User Think Time:** Think Time represents the time a user spends before submitting a request/transaction and has notation Z [17] [24].
- **Queue Length**: Service center Queue Length identifies a number of customers waiting for service in a specific queuing center and has notation Q_i [17] [24].

A Queuing Network can be open, closed, or mixed depending on its customer classes [13]:

- **Open Model:** A Queuing Network is open if customers/transactions arrive from outside of the system. Arrival rate (λ_i) denotes arrival intensity.
- **Closed Model:** A Queuing Network is closed if customers/transactions are generated inside the system (such as in a terminal system). In closed classes, the number of class c terminals (N_c) and think time for each class c terminal denote workload intensity (Z_c).
- **Mixed Model**: Systems that have both open classes (external input) and closed classes [13].

2.4 Queuing Network Simulation

Simulation modeling is becoming an increasingly popular method for network performance analysis. Generally, there are two forms of network simulation [13]:

- Analytical Modeling
- Computer simulation

Analytical modeling is conducted by a mathematical analysis that characterizes a network as a set of equations. The main disadvantage is its overly simplistic view of the network and inability to simulate the dynamic nature of a network. Thus, the study of a complex system always requires a discrete event simulation package, which can compute the time that would be associated with real events in a real-life situation. A software simulator is a valuable tool, especially for today's network with complex architectures and topologies. Designers can test their new ideas and carry out performance related studies, thus freeing themselves from the burden of "trial and error" hardware implementations.

A typical network simulator can provide the programmer with the abstraction of multiple threads of control and inter-thread communication. Functions and protocols are described either by finite-state machine, native programming code, or a combination of the two. A simulator typically comes with a set of predefined modules and a user-friendly GUI. Some network simulators even provide extensive support for visualization and animation [13]. In this thesis the JMT tool [4] has been used to model and simulate the queuing network with Finite Capacity Region presented in Chapter 5.

2.5 Summary

Most of the time processor resources are underutilized due to the reasons discussed in the chapter. Low utilization leaves a large number of execution resources idle each cycle.

In this chapter we provided theoretical background for SMT and HT technologies and showed in theory that they can efficiently deal with stated above problem. We also showed that queuing networks is a powerful yet simple modeling instrument allowing representation of complex system by restricting modeling aspects only for truly relevant system details and omitting irrelevant ones. This valuable advantage of QN permits us to restrict our attention only on modeling of the processor subsystem and omitting direct representation of other system subparts, like disks or memory.

Chapter 3

State of Art in SMT/HTT modeling

Current chapter presents two state of art QN models aimed at modeling SMT and HT technologies and some extensions for them.

3.1 SMT/HTT Model 1: Queuing Network Model

Authors from BMC Software propose a simple Queuing Network model [6] aimed at modeling a Hyper – Threaded processor architecture with a single core. They base their model on the fact that a hyper-threaded processor duplicates just some parts of the hardware (supporting extra architectural state), adding less than five percent to the chip size on the path from memory to CPU, but not the CPU itself [6].

This architecture allows for some parallel processing except at the actual instruction execution phase. That keeps the CPU itself busier at the moments of stalls, increasing the actual instruction execution rate [7]. Of course in order to take advantage of HTT, as with any multiprocessor system, multiple threads (in one or several processes) must simultaneously want access to a processor.

From the application's point of view each job visits one of the two logical processors offered to it by the operating system.

On the chip, each instruction execution goes through two phases. On the first one, an instance of the duplicated part of the hardware does preparatory work, looking up data and instructions in the cache or getting them from memory. And then the single real CPU executes the instruction.

The situation is modeled thus with three queues shown in Figure 3.1. Jobs arrive to the system at the rate of λ per second and go to one of the two preparatory queues with probabilities *p1* and *p2*, respectively. Since each job goes someplace, sum of *p1* and *p2* must be equal to one. Under assumption that OS has load balancing mechanism we can assume that *p1* = *p2* = 1/2.

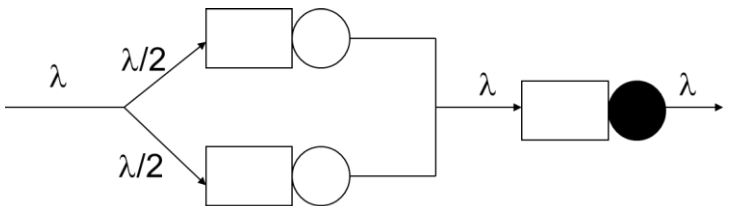

Figure 3.1: Queuing network model of a single core processor with HTT and load balancing

On the Figure 3.1, the solid circle represents the single physical CPU core, and open circles are the places where the work is done prior to the instruction execution.

Supposing that service time at the preparatory queue is *s1* seconds/job and the service time at the real CPU core is *s2* seconds/job and that the job inter-arrival times and service times are exponentially distributed, the mean response time of an open Queuing Network models becomes [6]:

(3.1)

$$R = \frac{p1 * s1}{1 - p1 * \lambda * \mathrm{s1}} + \frac{p2 * s1}{1 - p2 * \lambda * \mathrm{s2}} + \frac{s2}{1 - \lambda * \mathrm{s2}}$$

With load balancing assumption p1 = p2 = 1/2, Response time (R) becomes:

(3.2)

$$R = \frac{s1}{1 - \frac{\lambda}{2} s1} + \frac{s2}{1 - \lambda * s2}$$

In order to use model to predict system response time for real measured workload values λ, the parameters *s1* and *s2* need to be estimated. However there is a problem with parameters estimation, since the average total service time for each job, seen from the outside, can be measured but that does not tell anything about the internal service times, except that:

(3.3)

$$p1 * \lambda * s2 < 1,$$

$$p2 * \lambda * s2 < 1,$$

$$\lambda * s2 < 1$$

To validate the model authors ran a bon a single processor machine running at 2400 MHz Intel PowerEdge CPU. As a benchmark they use a self-made multithreaded Java application configurable to generate various kinds of compute intensive workloads.

In order to test the model and to understand Hyper – Threading architecture, authors found the best fitting values of *s1* and *s2* for analyzed dataset. Those values turned out to be equal to *s1 = 0.13* and *s2 = 0.81* for the model on Figure 3.2. These values suggest that about 15% of the work occurs in the parallel “preparatory phase” of the computation, while instruction execution consumes about 65% [6]. Figure 3.2 show the results of the experiment and analysis.

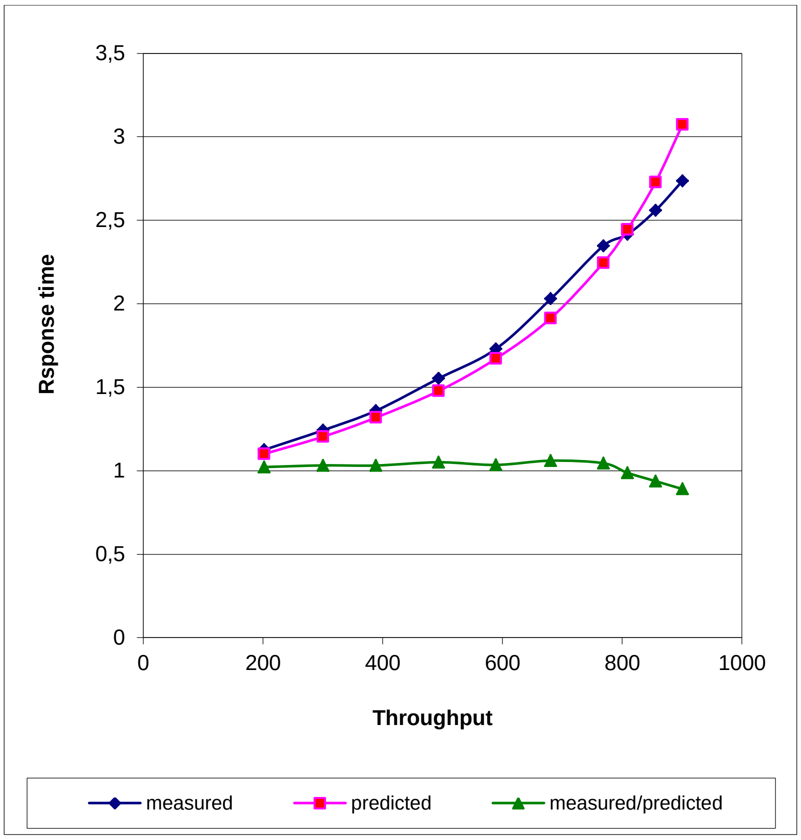

Figure 3.2: Response Time curves for one-second job (single processor, HT on, predictions for model from Figure 3.1)

The fact that the measured data is fitted so well with reasonable parameter values suggests that the model is sound. Following section extends this model to take into consideration also dual core and quad core processors, that is, processors with two and four cores, respectively.

3.1.1 Model Extension

Unfortunately authors of the work do not extend their research on the technology for different processor types and propose model only for single core processor.

In our research, instead, we try to evaluate SMT and HTT performance effects for different system configurations, which certainly will result in more realist case usages and produce more comprehensive assessment of the technologies. Under changed system configuration we intend a different number of active CPU cores and different processor clock frequency. All examined configurations are listed in detail in “Methodology” section of next chapter.

Finally, we tried to extend proposed model to reflect also system configurations examined in our work.

3.1.1.1 Extension 1: Dual core processor with HTT

Figure 3.3 illustrates an extension of previous model for dual core processor supporting HTT, or equivalently two-way SMT. The model is obtained simply by replication of the processor core, including all its instruction preparation and execution resources.

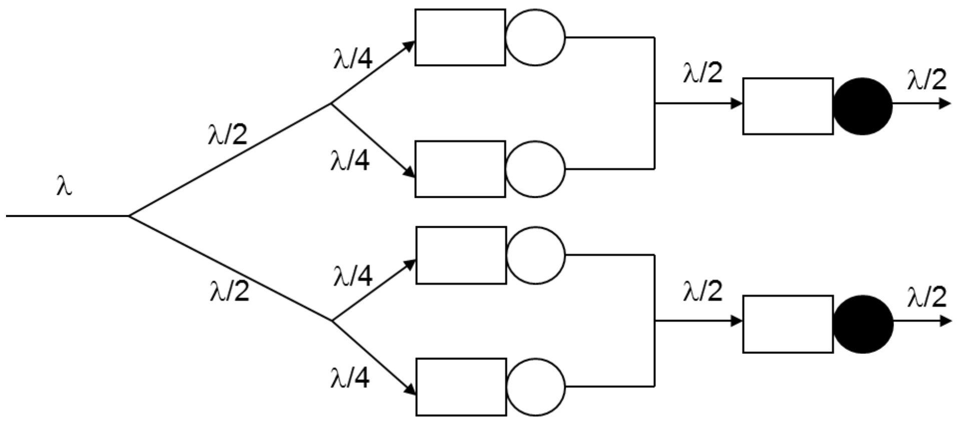

Figure 3.3: QN model for Dual core processor with HTT and optimal load balancing mechanism

On the Figure 3.3, the solid circles represent the execution units, and open circles are the places where the work is done prior to the instruction execution.

From the application’s point of view, now with active HTT each job visits one of the four logical cores offered to it by the operating system. As in previous case, on chip, each instruction execution goes through two phases. First, an instance of the duplicated part of the hardware inside one of two cores does preparatory work, looking up data and instructions in the cache or getting them from memory; and then the single real CPU executes the instruction within the core.

The situation is modeled with six queues entering the service centers shown on the Figure 3.3, with three queues belonging to one core and remainder queues to another. Assuming perfect load balancing, jobs arriving to the system at the rate of λ per second then are directed to one of the two cores with equal probabilities, where they are further distributed to preparatory queues again with equal probabilities. Since each job goes someplace, the sum over all probabilities results to be *one*.

Again supposing that service time in four preparatory queues is *s1* seconds/job and the service time at queues representing execution units is *s2* seconds/job and that the job inter-arrival times and service times are exponentially distributed, the mean system response time of an open Queuing Network model from Figure 3.3 becomes:

$$R = \frac{p1 * s1}{1 - p1 * \lambda * s1} + \frac{p2 * s1}{1 - p2 * \lambda * s2} + \frac{p3 * s1}{1 - p3 * \lambda * s1} + \frac{p4 * s1}{1 - p4 * \lambda * s1} + \frac{(p1 + p2) * s2}{1 - (p1 + p2) * s2} + \frac{(p3 + p4) * s2}{1 - (p3 + p4) * s2} \tag{3.4}$$

With load balancing both between and inside cores ($p1 = p2 = p3 = p4 = \lambda/4$) the Response time becomes:

$$R = \frac{s1}{1 - \frac{\lambda}{4} * s1} + \frac{s2}{1 - \frac{\lambda}{2} * s2} \tag{3.5}$$

In order to use model to predict system response time for real measured workload values λ, the parameters *s1* and *s2* need to be estimated. Unfortunately, authors of the original model do not describe the way they estimated values of parameters *s1* and *s2*, but just mention that they are the best fitting values.

In our case, optimal values estimation we use two-phase process based on error minimization. Under error we intend prediction error between dataset and model. First, we feed the model with feasible *s1, s2* parameter values found from the following set of equations:

(3.6)

$$\frac{1}{4} * \lambda * s1 < 1,$$

$$\frac{1}{2} * \lambda * s2 < 1,$$

Obtained in such way values of s1 and s2 are used to parameterize model, for which is then error estimated. Subsequently, this error is minimized using GRG Nonlinear solver, embedded in our tool used for processing, which always convergence to optimal values for all smooth nonlinear problems, which is also our case. Excessively, to ensure convergence to optimal values, another solving method – Evolutionary engine is applied. Obtained values are guaranteed to be optimal, thus producing lowest possible model prediction error.

3.1.2 Extension 2: Quad core processor with HTT

Figure 3.4 illustrates an extension for quad core processor supporting HTT. The model is again obtained by duplication of the processor core four times. The model characteristics remain the same as in case with Extension 1, but now OS recognizes eight virtual cores with active HTT.

Again assuming that operating system has working load balancing mechanism, the probability of request being processed by any of eight virtual cores is equal 1/8, thus the mean system response time of an open Queuing Network model from Figure 3.4 becomes:

$$R = \sum_{i=1}^{8} \frac{p_i * s1}{1 - p_i * \lambda * s1} + \sum_{i=1}^{4} \frac{\frac{\lambda}{4} * s2}{1 - \frac{\lambda}{4} * s2} \qquad (3.6)$$

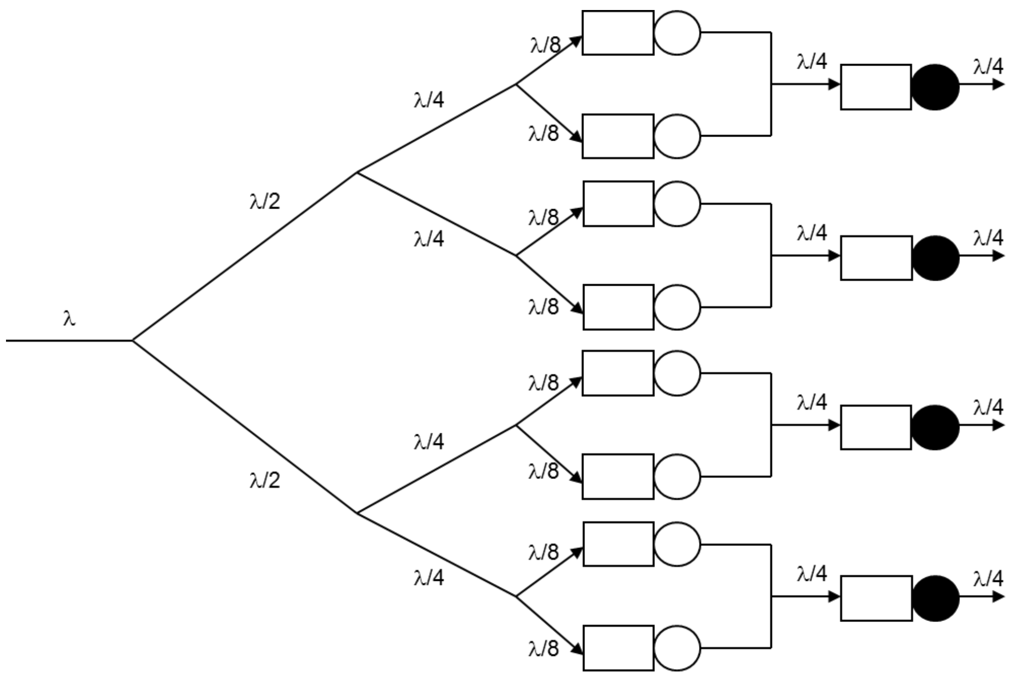

Figure 3.4: QN model for Quad core processor with
HTT and optimal load balancing mechanism

With load balancing both between cores and inside core ($p1 = p2 = \cdots = p8 = \lambda/8$)

$$R = \frac{s1}{1 - \frac{\lambda}{8} * \mathrm{s1}} + \frac{s2}{1 - \frac{\lambda}{4} * \mathrm{s2}} \tag{3.7}$$

In order to use model to predict system response time for real measured workload values λ, the parameters *s1* and *s2* need to be estimated. The methodology remains the same as for Extension 1, with the only difference that initial (border) values of parameters s1 and s2 are obtained from the following equations:

(3.8)

$$\frac{1}{8} * \lambda * \mathrm{s1} < 1,$$

$$\frac{1}{4} * \lambda * \mathrm{s2} < 1,$$

Again, obtained parameter values are guaranteed to be optimal, thus producing lowest possible model prediction error.

3.2 SMT/HTT Model 2: Birth-Death Markov chain Model

3.2.1 Introduction

Author of the second model propose a Birth-Death Markov model targeted at modeling SMT enabled single core processor architecture having up to eight threads [8].

The key idea underlying is that SMT processor is best modeled as a load-dependent service center, while superscalar processor is easily modeled as simple service queue [8]. The same observation we use in our birth-death Markov chain model developed in Chapter 6, however with respect to this model we propose simpler model based only on two parameters μ and α. The model we propose can be considered as a special case of this model.

A Markov chain model allows solving for system performance metrics derived from a probability distribution (produced by the model) on the queue population. This provides more accurate results for a load-dependent system such as the SMT processor than simpler queuing models that depend only on the average population in the queue.

Ordinary superscalar processor (superscalar represents the superscalar processor unmodified for SMT execution) is the M/M/1 queue, whose Markov model is shown in Figure 3.5.

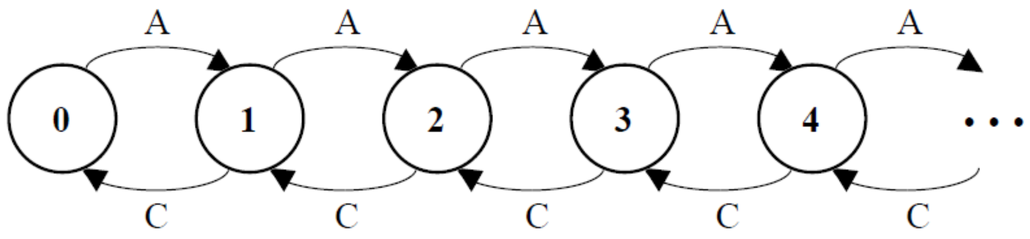

Figure 3.5: The Markov state diagram for a single superscalar processor

The numbers in the circles are the population in the queue; the arcs flowing right are the rate at which jobs enter the system, and the arcs flowing left are the rate at which jobs leave the system.

The arrival rates for the superscalar processor on Figure 3.5 (λ_i) are all equal to a constant arrival rate ($\lambda_i = A$). Because the throughput of the system is independent of the number of jobs in the system, the completion rates (μ_i) are all equal to a constant completion rate, ($\mu_i = C$). The completion rate is assumed to be that of the unmodified superscalar processor found previously in their work.

A Markov system can be solved for the equilibrium condition by finding population probabilities that equalize the flow in and out of every population state.

Known values A and C, or for the most quantities $q = A/C$, the well-known solution for M/M/1 system can be used.

The probability (p_k) that k jobs are in the queue:

(3.8)

$$p_k = p_0 \left(\frac{A}{\mathrm{C}}\right)^k = p_0 p^k$$

Allows solving that zero jobs are in the queue:

(3.9)

$$p_0 = \frac{1}{1 + \sum_1^{inf} q^k} = 1 - q$$

Thus the utilization (U) is the probability that there is more than one job in the system, and it is simply:

(3.10)

$$U = 1 - p_0 = q = \frac{A}{C}$$

The average population (N) of the queue is:

(3.11)

$$\bar{N} = \sum_{k=0}^{inf} k\, p_k = \frac{q}{1-q}$$

And the response time (R), from Little's Law [17]:

(3.12)

$$R = \frac{\bar{N}}{\lambda} = \frac{1/C}{1-q}$$

3.2.2 SMT processor model

Similar, but more complex Markov model for the SMT processor is shown on Figure 3.6, aimed at modeling a limited load-dependent server.

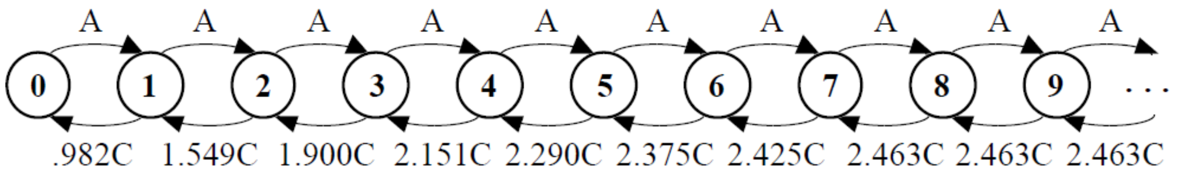

Figure 3.6: The Markov state diagram for single core eight-way SMT processor

It is similar to the M/M/m queue - a queue with multiple equivalent servers, however, in the M/M/m queue the completion rates vary linearly with the queue population when the population is less than m, which is not true here. In this system on Figure 3.6, the completion rates are the throughput rates of non-modified superscalar processor ($\mu_i = C$) multiplied by the factor which depends on queue population. The completion rate figures have been obtained by the authors after simulation performed on SMT processor.

For example, when there is only one job in the system, the SMT processor runs at 0.98 times the speed of the superscalar, but when five jobs are in the system, it completes jobs at 2.29 times the speed of the superscalar. That is, while the job at the head of the queue may be running more slowly, because all five jobs are actually executing the completion rate is higher, proportional to the throughput increase over the superscalar.

Another assumption - the rate of completion is constant once there are more jobs than thread slots in a system. This assumes that the cost of occasionally swapping a thread out of the running set (an operating system context switch) is negligible. This is the same assumption to use a single completion rate for the superscalar model. This assumption favors the superscalar, because it incurs the cost much earlier (when there are two jobs in the running set).

Because the arrival rate is still constant, but the completion rates (μ_i) are not, the formula for the probability of population k is:

$$p_k = p_0 \prod_{i=1}^{k} \frac{A}{\mu_i} \tag{3.13}$$

For the simulation results, this series becomes manageable because the tail of the Markov state diagram is once again an M/M/1 queue. So,

(3.14)

$$p_k = \begin{cases} p_0 \prod_{i=1}^{k} \frac{A}{\mu i}, & k \le 8 \\ p_8 (\frac{A}{2.463 * C})^{k-8} = p_0 \, (\frac{A}{C})^8 * \frac{1}{M} * \, (\frac{A}{2.463 * C})^{k-8}, & k > 8 \end{cases}$$

Where M = (0.982) (1.549) … (2.425) (2.453).

As with the M/M/1 queue, because all probabilities sum to one and we can express all probabilities in terms of p_0, we can solve for p_0, and thus the utilization.

We can solve it because the sum of all probabilities fork $k \geq 8$ is a series that reduces to a single term (as long as A/2.463C < 1), as does the M/M/1 queue when A/C < 1, thus:

(3.15)

$$p_0 = \frac{1}{1 + \frac{A}{C} * \frac{1}{0.982} + \left(\frac{A}{C}\right)^2 * \frac{1}{0.982 * 1.549} + \cdots + \left(\frac{A}{C}\right)^8 * \frac{1}{M} \frac{1}{1 - \frac{A}{2.463C}}}$$

In order to calculate response times, first we need to know the average population in the queue:

(3.16)

$$\overline{N} = \sum_{k=0}^{inf} kp_k = \sum_{k=0}^{7} kp_k + \sum_{k=8}^{inf} (k+8) p_8 q^{k'}$$

(3.17)

$$\overline{N} = \sum_{k=0}^{7} kp_k + p_8 \frac{q'}{(1-q')^2} + 8p_8 \frac{1}{(1-q')}$$

Response time can then be computed from the average population and the jobs arrival rate (λ) using Little's Law [17]:

(3.18)

$$R = \frac{\overline{N}}{\lambda}$$

Unfortunately, author of the model does not provide any validation approach. The validation results of our model, which can be considered as a special case of this model, are represented in Chapter 6.

Chapter 4

Potential of SMT / HTT

In this chapter we perform experimental validation of the SMT and HTT theoretical aspects discussed previously in Chapter 2. Also we perform a series of benchmark runs to reveal true performance impacts enabled by Simultaneous Multithreading and Hyper-Threading Technology for different system configurations using two types of benchmarking suite.

4.1 Performance Claims

The advantages of hyper-threading are listed as improved support for multi-threaded code, allowing multiple threads to run simultaneously; increased throughput and reduced response time. According to Intel, its first implementation only used five percent more die area on chip comparing to equivalent non-hyper threaded processor, but the performance increase was 15 – 30% greater [2][11]. For a previous generation processors Intel claims up to a 30% performance improvement compared with an otherwise identical, non-simultaneous multithreading processors [2][10][11].

According to [10], in some cases a previous generation Pentium IV processor running at 3 GHz clock frequency with HT activated can even beat a Pentium IV running at 3.6 GHz with HT turned off. Intel also claims significant performance improvements with a HTT in some artificial intelligence algorithms.

Technology improves throughput and efficiency for better performance and performance per watt. Intel HT Technology provides greater benefits in Intel Core i7 processors, and other processors based on the Nehalem core including the Xeon 5500 series of server processors than was possible in the Pentium 4 processor era when it was first introduced [12]. The discussion in [12] refers to Intel 64 architecture, with particular emphasis on Intel processors based on the Nehalem core including the Core i7 Processor and the Xeon 5500 series of processors. According [12] the performance improvement for this workload is 30% (1.30x) when Intel HT Technology is enabled for the workload consisting mostly of integer ALU operations. The response time as viewed by the client dropped from 50 ms per transaction to 37 ms per transaction with Intel HT Technology enabled (about 26%).

4.2 Test bed configuration

The system under test used for benchmarking has Intel Core i7 920 processor and 12 GB of DD3 DRAM installed. The processor is characterized by 4 cores, default clock speed of 2.66 GHz, HT Technology and three eight-byte DD3 channels. Processor cache memory has following characteristics [5]:

- 1st level instruction cache: 32 KB per core 4-ways
- 1st level data cache: 32 KB per core 8-ways (write-back)
- 2nd level data cache: 256 KB per core
- 3d level data cache: 8 MB per four cores shared across all cores

4.3 Methodology

To achieve a fairly comprehensive assessment of SMT/HTT, the evaluation has been performed for different system configurations. Since the key component of the system, which we try to model, is a processor, using a small self-made java program we were able to alter processor states. In total, performance impact of Intel’s HTT has been evaluated for all the following system configurations:

- 4 cores, HT enabled, 2660 MHz
- 4 cores, HT enabled, 1596 MHz
- 2 cores, HT enabled, 2660 MHz
- 2 cores, HT enabled, 1596 MHz
- 1 core, HT enabled, 2660 MHz
- 1 core, HT enabled, 1596 MHz

2660 MHz is a maximum reachable clock frequency that a processor can have, and 1596 MHz is a lowest frequency that can be set for a processor. The choice of “border” frequencies is not accidental and it is intended to reveal the most productive technology usage cases and to understand how drastic changes in the processor clock frequency affects the performance gain enabled by SMT/HTT technology. Additionally, the configurations listed above represent of the most encountered computer systems.

4.3.1 Obtain system measures

Next, for each examined system configuration, representing changed processor state, the system needs to be measured under certain workloads to obtain *workload* and *performance measures*.

Subsequently, in Chapter 6, this data will serve as an input for SMT/HTT models evaluation step and obtained model performance indices will be compared to the measured system outputs.

4.3.2 Benchmarking

Benchmarks are used to generate workload and record both workload and performance metrics. In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of a system, normally by running a number of standard tests and trials against it, accordingly, the workload and performance measures are strictly dependent on the type of benchmark and on each benchmark run or trial itself. In our research we use two different benchmarking suites to assess SMT/HTT performance, each having its own measuring approach and features.

4.3.3 SPEC Power SSJ 2008

First benchmark used for evaluations is SPEC Power SSJ 2008. It is a product developed by the Standard Performance Evaluation Corporation (SPEC) and it is the first industry standard for measuring both performance and power consumption of computer systems [18].

SPEC benchmark allows estimation of such performance measures, like *system throughput* and *processor utilization*, but unfortunately does not provide any suitable means to obtain *system response time.*

At a high level benchmark models a server application with a large number of users [19]. Requests from these users arrive at random intervals (modeled with a negative exponential distribution), and processed by a finite set of threads of the server. The exponential distribution may result in bursts of activity; during this time, requests may queue up while other requests are being processed. The system will continue processing transactions as long as there are requests in the queue.

The whole benchmark suite consists of three main components [18] [20]:

- **Server Side Java (SSJ) Workload:** SSJ Workload is a Java program designed to exercise the CPU(s), caches, memory, the scalability of shared memory processors, JVM (Java Virtual Machine) implementations, JIT (Just In Time) compilers, garbage collection, and other aspects of the operating system of the system under test (SUT).

- **Power and Temperature Daemon (PTDaemon):** The PTDaemon is to offload the work of controlling a power analyzer or temperature sensor during measurement intervals to a system other than the SUT.
- **Control and Collect System (CCS)**: CCS is a multi-threaded Java application that controls and enables the coordinated collection of data from multiple data sources such as a workload running on a separate SUT, a power analyzer, and a temperature sensor. It also includes Visual Activity Monitor (VAM) - software package designed to display activity from one, two, or three SUT"s simultaneously, in combination with the SPECpower_ssj2008 benchmark.

In experiments the whole system has been configured without PTDaemon component because of non-availability of the required equipment, and secondly, because eventually the goal is to understand performance and not power impact enabled by the technology.

Figure 4.1 illustrates the architecture of a standard SPECpower_ssj2008 benchmark implementation including Power Analyzer and Temperature sensor components, which in our case are omitted for the reasons listed above.

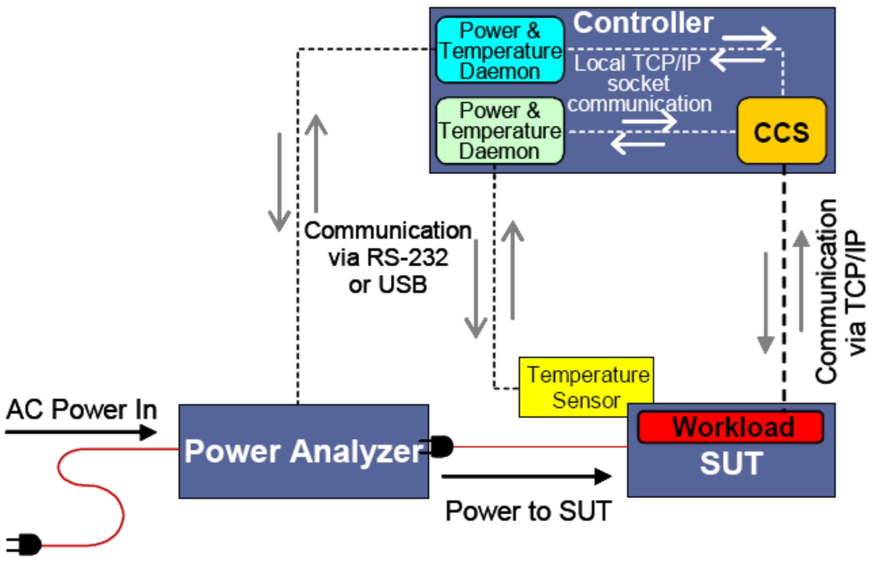
Controller
Power & Temperature Daemon
Local TCP/IP socket communication
Power & Temperature Daemon
CCS
Communication via RS-232 or USB
Communication via TCP/IP
Temperature Sensor
AC Power In
Power Analyzer
Workload
SUT
Power to SUT

Figure 4.1: Architecture of a s SPEC Power SSJ 2008 benchmark implementation

4.3.3.1 SPEC’s Server Side Java (SSJ) Workload

The SSJ Workload is a component responsible for benchmark workload generation. First, the system needs to be calibrated. The calibration phase is used to determine the maximum throughput that a system is capable of sustaining. Once this calibrated throughput is established, the system runs at a series of target loads. Each load runs at some percentage of the calibrated throughput. By default, 3 calibration intervals are used, each four minutes long. The calibrated throughput (used to calculate the target throughputs during the target load measurements) is calculated as the average of the last 2 calibration intervals.

For compliant runs, the sequence of load levels decreases from 100% to 0% in increments of 10%. At each load level the system is measured for 120 seconds. The intervals between load level measurements, the so-called “warm-up” and “cool-down” are equal to 30 seconds. Measuring the points in decreasing order limits the change in load to 10% at each level, resulting in more stable performance measurements. Using increasing order would have resulted in a jump from 100% to 10% moving from the final calibration interval to the first target load, and another jump from 100% to Active Idle at the end of the run.

4.3.4 A synthetic benchmark

On the system under test (SUT) we run a simple web application written in Java (a *.war file) deployed on Apache Tomcat webserver. This application is composed of a single page which performs a computation on floating point numbers and stresses both the CPU and memory hierarchy. The service time of this page is exponentially distributed.

To load the system we use a multi-threaded load generator that generates http requests with exponentially distributed inter-arrival times. The load generator runs on a separate machine connected with the system under test by 100 MBits/s network connection to avoid network bottlenecks.

The advantage of using the load generator and a synthetic workload is that we can create a load which can be easily modeled, e.g., exponential inter-arrival times and exponential service-times. Moreover, the load generator besides processor utilization can also measure system response time, so that we can analyze not only the utilization and throughput, but also the response time.

4.3.4.1 Synthetic Workload

The crucial points of workload can be listed as:

- Time complexity for the processing of one request should be O(n)
- The workload should stress the memory hierarchy
- Partially memory and partially CPU bound
- Possibility to tune impact of memory accesses

The workload consists of two phases: initialization phase and actual execution phase. At initialization phase a working set of $\boldsymbol{d}$ vectors and a filter vector $\boldsymbol{f}$ of length $\boldsymbol{h}$ are generated. Each vector has a random length uniformly distributed between $\boldsymbol{cMin}$ and $\boldsymbol{cMax}$.

At execution phase a random vector $\boldsymbol{v}$ is selected and two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of length $\boldsymbol{n}$ are allocated. Then vector $\boldsymbol{a}$ is filled with sums of $\boldsymbol{p}$ random strides of length $\boldsymbol{w}$ from $\boldsymbol{v}$ and convolved with filter vector $\boldsymbol{f}$. Convolution result is stored in $\boldsymbol{b}$. Finally the sum of the elements in $\boldsymbol{b}$ is returned.

Summarizing, execution phase consists of three main steps having complexity O(n):

1. Filling the source vector **a**
2. Compute **b** performing the convolution of **a** and **f**
3. Sum the elements of **b**

There are some cache size considerations which must be met for optimal workload generation. The memory used by the working set vectors should exceed the size of the LLC (Last Level Cache), that is to say:

$$d * 8 * \left(cMin + \frac{(cMax - cMin)}{2} \right) >> LLC_{size} \tag{4.1}$$

Source and destination vectors a and b should fit in a private cache L1 or L2:

$$n * 2 * 8 << LX_{size} \tag{4.2}$$

Filter vector ***f*** must fit in the first level cache:

$$h * 8 << L1D_{size} \tag{4.3}$$

Figure 4.2 demonstrates the process of filling the source vector ***a*** with number of strides $\boldsymbol{p} = \mathbf{3}$ each having length $\boldsymbol{w} = \mathbf{2}$**:**

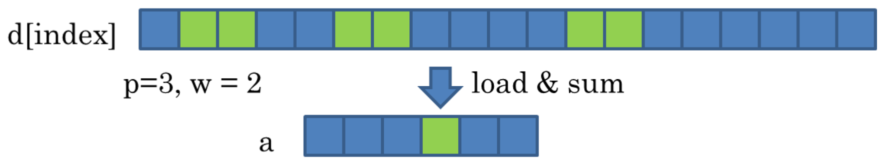

Figure 4.2: Synthetic workload: Filling source vector ***a***

For small ***w***, a full cache line is read but not used; the prefetcher helps only for large ***w.***

Figure 4.3 demonstrates the convolution process for obtained source vector ***a*** with filter vector ***f***; the results is stored in destination vector ***b***:

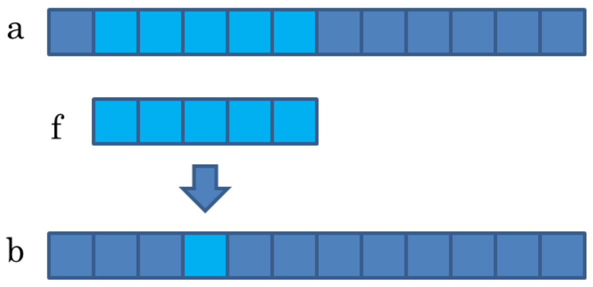

Figure 4.3: Synthetic workload: Convolution process between source vector ***a*** and filter vector ***f***

Finally, Figure 4.4 how the result value is obtained:

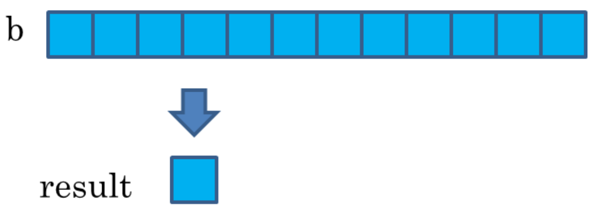

Figure 4.4: Synthetic workload: Result calculation

In our benchmark runs to generate optimal workload the values of ***p***, ***w*** and ***n*** have been defined adhering to defined above observations and are equal to 20, 32 and 10000, respectively.

4.3.4.2 IronLoad Generator

IronLoad generator is used to generate benchmark workload and able of capturing such performance measures like processor *utilization* and system *response time*. For each system configuration listed in Section 4.3 the benchmark run is performed. Each benchmark run consists of two main phases: maximum throughput estimation that system is capable of sustaining for closed loop and run at series of target loads. Each load runs at some percentage of target utilization. If utilization is I, then target load is equal to:

$$TargetX = \frac{maxX}{100} * i \tag{4.4}$$

For compliant runs the sequence of target utilization load levels decreases from 80% to 5% in increment of 5%, in total resulting in 16 load levels. Load level duration is set to be 1200 seconds and interval between load levels is equal 30 seconds. All the measured data is saved in CSV format.

4.4 Performance results

The following performance data have been obtained after a series of benchmark runs for S*ystem Throughput*, P*rocessor Utilization* and S*ystem Response Time.*

The Response Time measure is available only for synthetic benchmark, since SPEC Power SSJ 2008 does not provide ant suitable means get this measure. Feasible values for Utilization vary in range from 0 to 100%.

Each plot illustrates performance data obtained at specific CPU clock frequency for all examined combination of cores.

4.4.1 System Throughput

System Throughput is estimated as number of operations performed in a second. The measurement results are available for both synthetic and SPEC Power benchmarks.

Estimated performance gain enabled by HTT over the equivalent system configuration but without HTT, is indicated in the right upper corner of each plot and it is strictly related to the value of parameter Alpha, calculation of which is discussed in the corresponding section of next chapter: $Gain = Alpha * 100$

4.4.1.1 SPEC Power SSJ 2008 benchmark

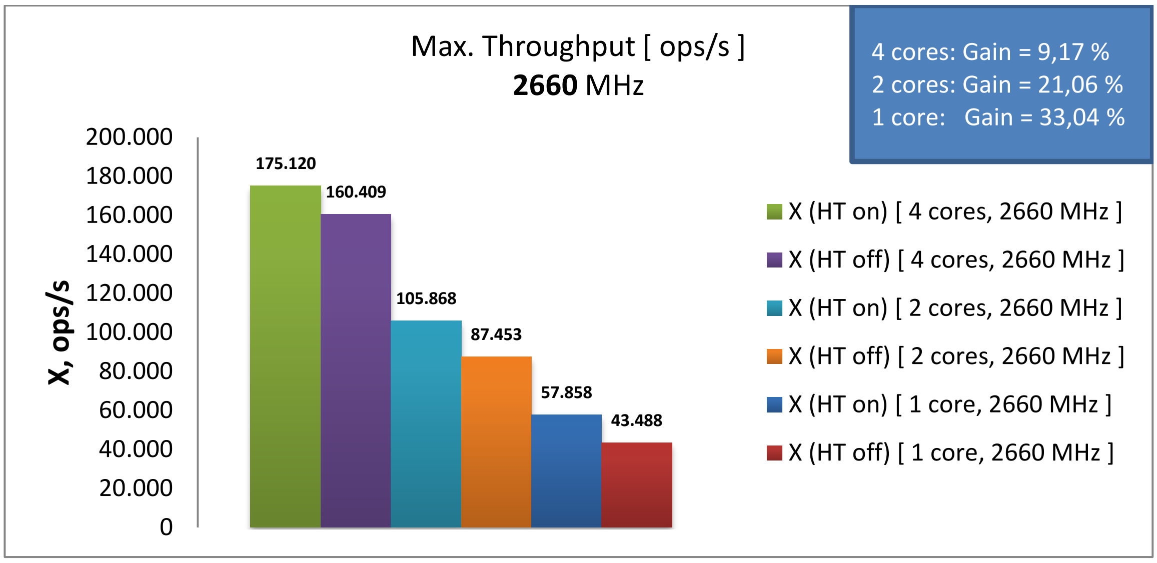

Figure 4.5: System Throughput with HT {on, off} for
{4, 2, 1} cores at 2660 MHz

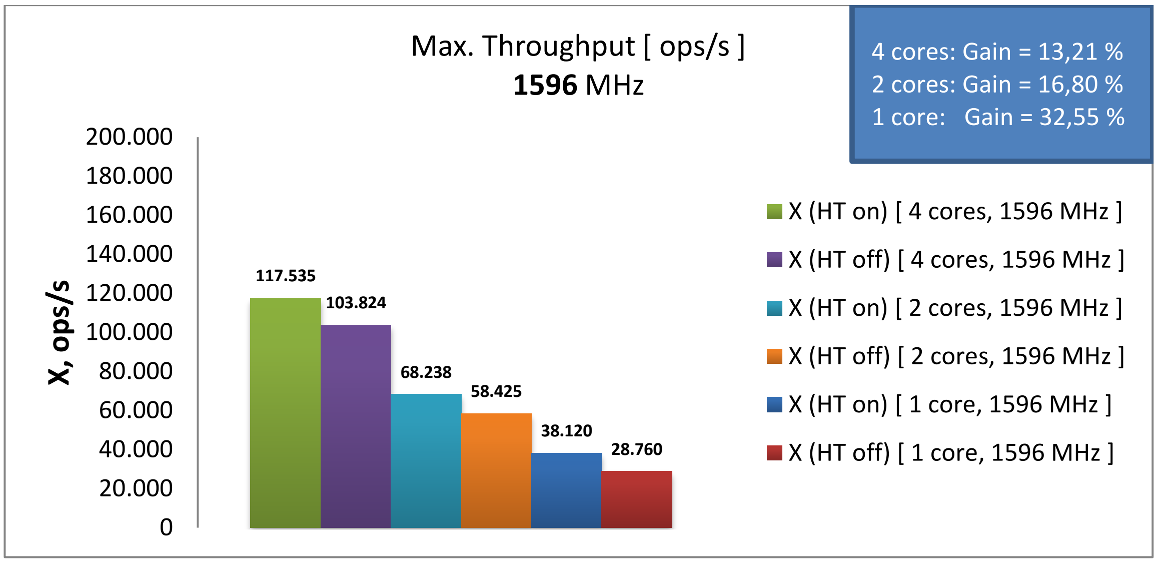

Figure 4.6: System Throughput with HT {on, off} for
{4, 2, 1} cores at 1596 MHz

SPEC Power benchmark exhibits significantly higher performance increase range. Depending on the configuration, throughput increase varies from 9 to 33%. Again, greater increment is achieved at “weaker” configurations.

4.4.1.2 Synthetic benchmark

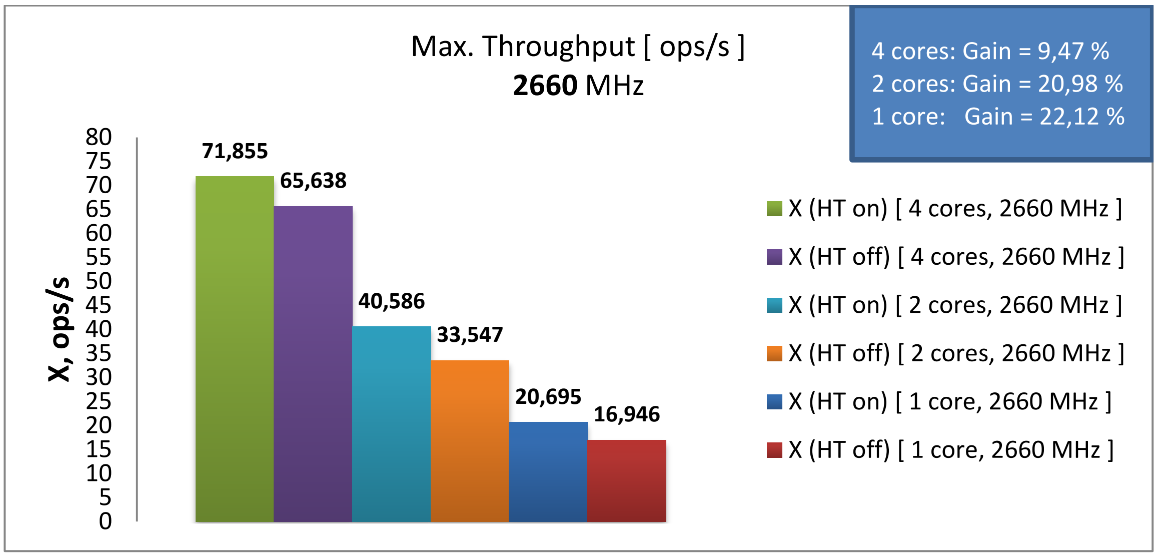

Figure 4.7: System Throughput with HT {on, off} for {4, 2, 1} cores at 2660 MHz

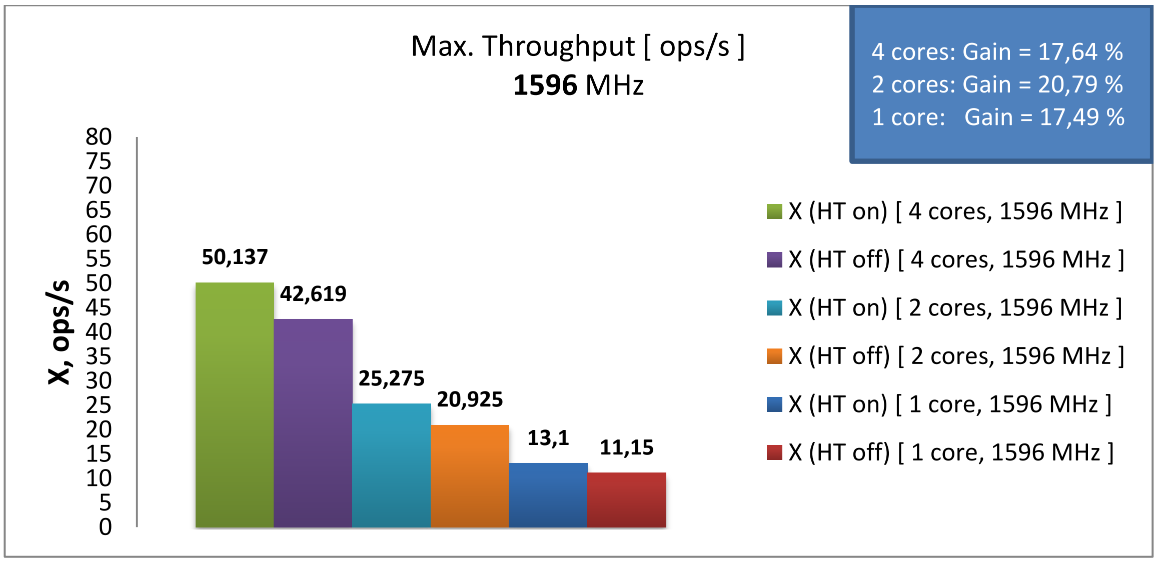

Figure 4.8: System Throughput with HT {on, off} for {4, 2, 1} cores at 1596 MHz

Experiment data reveals that for synthetic benchmark throughput gain varies from 9 to 22% depending on the configuration of the system. Interestingly, gain ration is preserved for different frequencies, except for configuration with 4 cores. Maximum gain (in relative terms) is achieved at “weaker” configurations.

4.4.2 Processor Utilization

Measurement of processor utilization can be very misleading. Generally, multi-core systems present a reporting dilemma to the measurement community [6]. For example, when both processors on a dual processor machine are active during an interval is the utilization 100% or 200%? To eliminate the ambiguity, the number of processors for which the utilization is reported has to be specified and a performance measurement tool must use one convention consistently and make clear how its output is to be interpreted.

For a hyper-threaded processor, which is a single physical CPU designed to appear to the operating system and its measurement tools as two separate cores, the problem is even more compounded. Since hyper-threading is in principle transparent to the operating system and applications, existing measurement tools need no modifications to run in a hyper-threaded environment [6]. But what will they report for processor Utilization – the answers vary.

Traditional way is to define utilization as a fraction of utilized threads. In the case with HTT on number of available to the system threads doubles, hence resulting in lower utilization.

Below there is a series of plots for utilization measured for different system configurations using SPEC Power SSJ 2008 benchmark and a synthetic benchmark.

As a reader can notice from Figures 4.10 – 4.13 and figure from Motivation section, utilization curves with HTT off are straight lines for all system configurations. This fact can be easily described by Utilization law [17][24], which represents linear relationship between utilization and workload:

$$U_{CPU} = X * d_{CPU,} \tag{4.1}$$

where d_{CPU} is a service demand. In fact, for configurations with HTT off, according to plots utilization scales linearly with workload. However, the situation changes in the presence of SMT (Figures 4.10 – 4.13) and the growth becomes nonlinear.

There are several research papers focusing on service demand estimation under the assumption that Utilization law holds. As an example parameter d_{CPU} can be estimated on some data samples using Utilization law, by measuring U and X for some intervals. The resulting U versus X must be reported on a chart. Since we have linear dependence we can find the line that better approximates obtained U versus X samples using Linear Regression estimation. The slope of the line would represent service demand (d_{CPU}).

Figure 4.4 illustrates example of d_{CPU} calculation for the SPEC Power dataset. In example below $d_{CPU} = 0{,}0005.T$erminology and notation details can be found in Chapter 2.

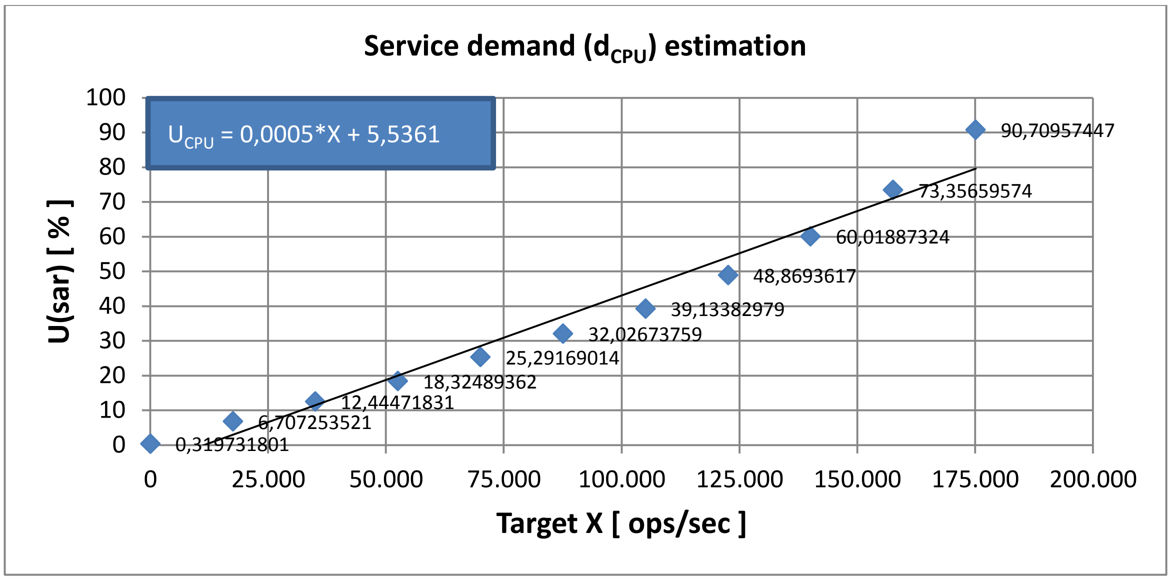

Figure 4.9: Service demand (d) estimation for SPEC Power benchmark data samples at 4 cores, 2660 MHz wit HT configuration

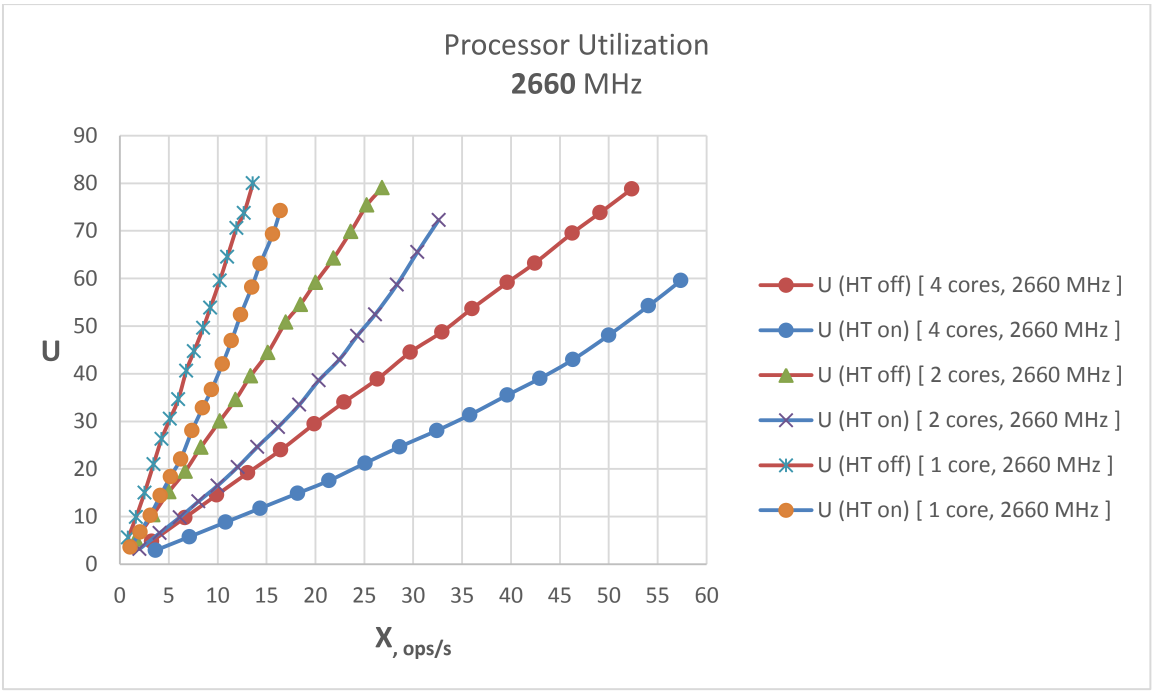

Figure 4.10: Synthetic benchmark: Processor Utilization with HT {on, off} for {4, 2, 1} cores at 2660 MHz

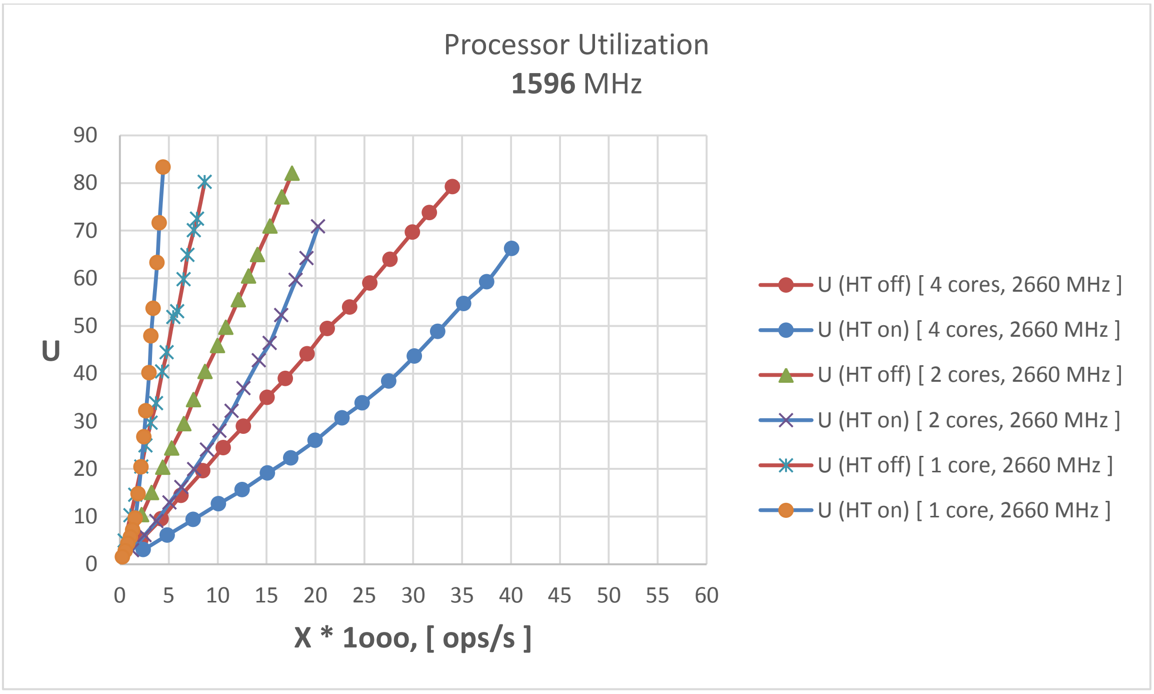

Figure 4.11: Synthetic benchmark: Processor Utilization with HT {on, off} for {4, 2, 1} cores at 1596 MHz

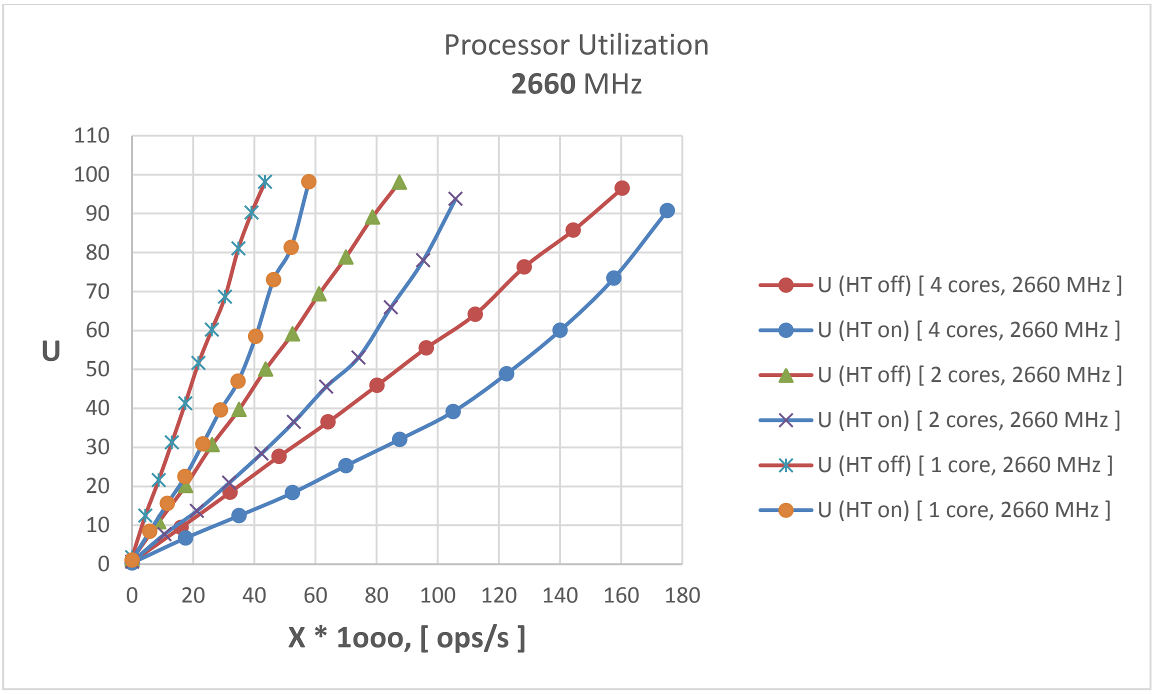

Figure 4.12: SPEC Power: Processor Utilization with HT {on, off} for {4, 2, 1} cores at 2660 MHz

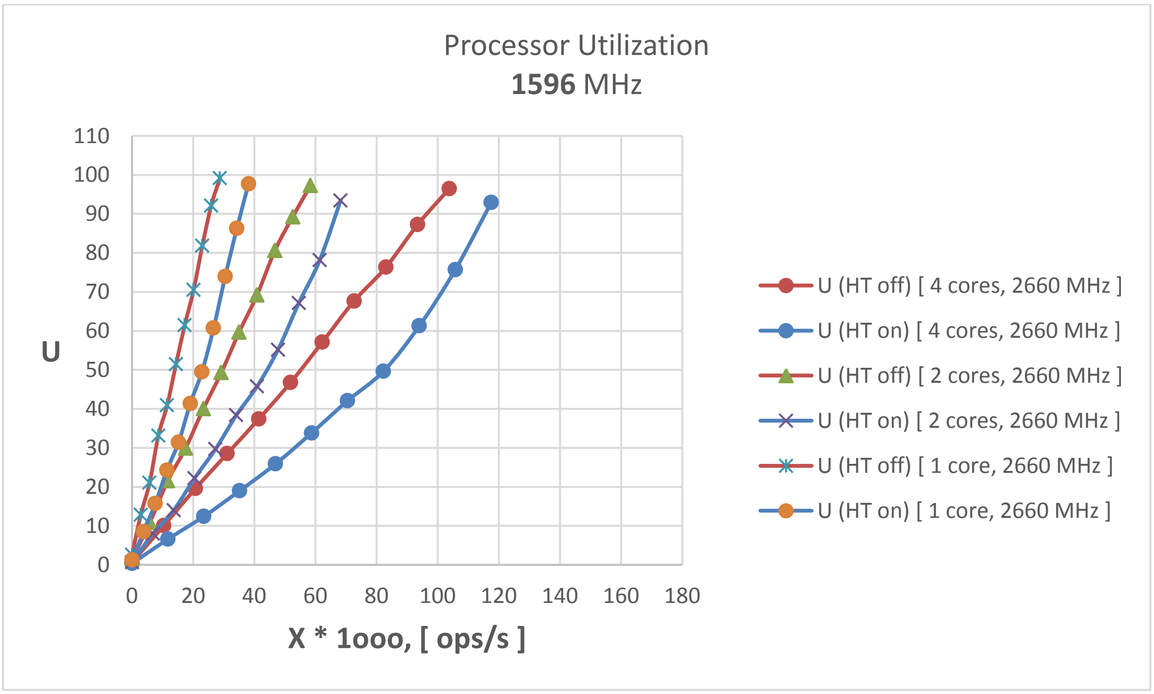

Figure 4.13: SPEC Power: Processor Utilization with HT {on, off} for {4, 2, 1} cores at 1596 MHz

Utilization curves on Figures 4.10 – 4.13 for configurations with HTT on are slightly convex. It means that Utilization law does not appropriately hold for the cases when HTT is on.

4.4.3 System Response Time

System Response time is important characteristic, which ultimately also identifies user waiting time. Response time is measures as elapsed time from the moment when the request has been submitted to the system and until it get serviced. Than lower this performance characteristic, that greater system performance. The figures below illustrate response time data of synthetic benchmark for different system configurations.

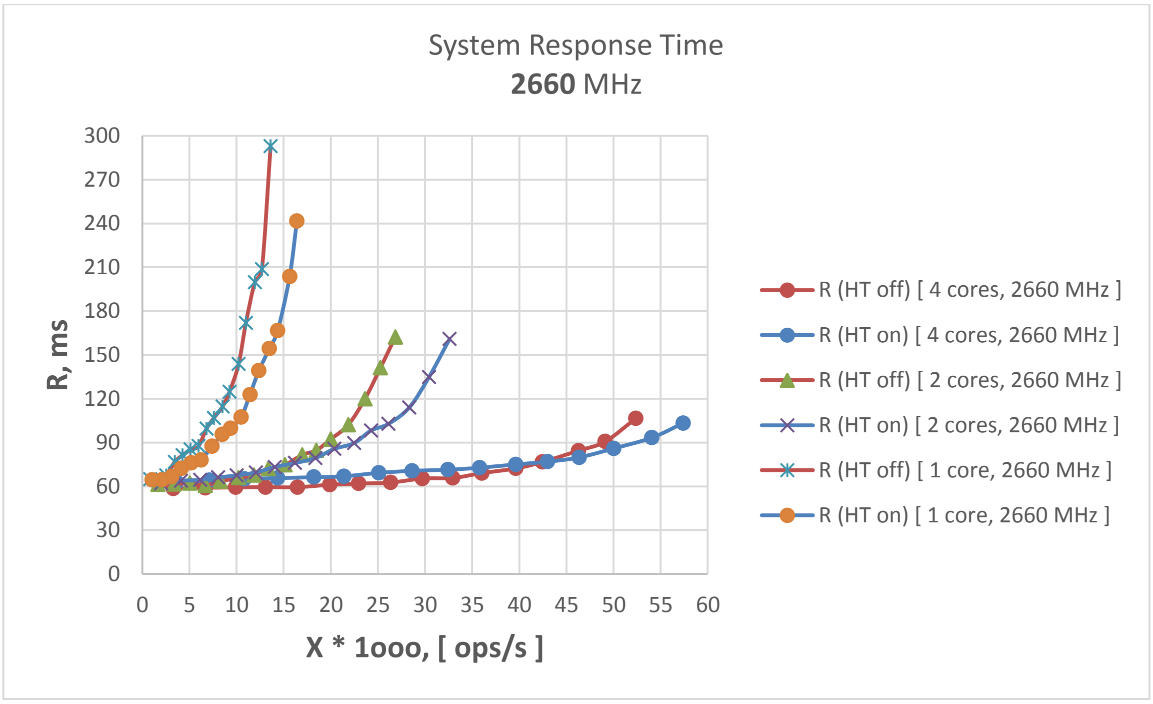

Figure 4.15: System Response Time with HT {on, off}
for {4, 2, 1} cores at 2660 MHz

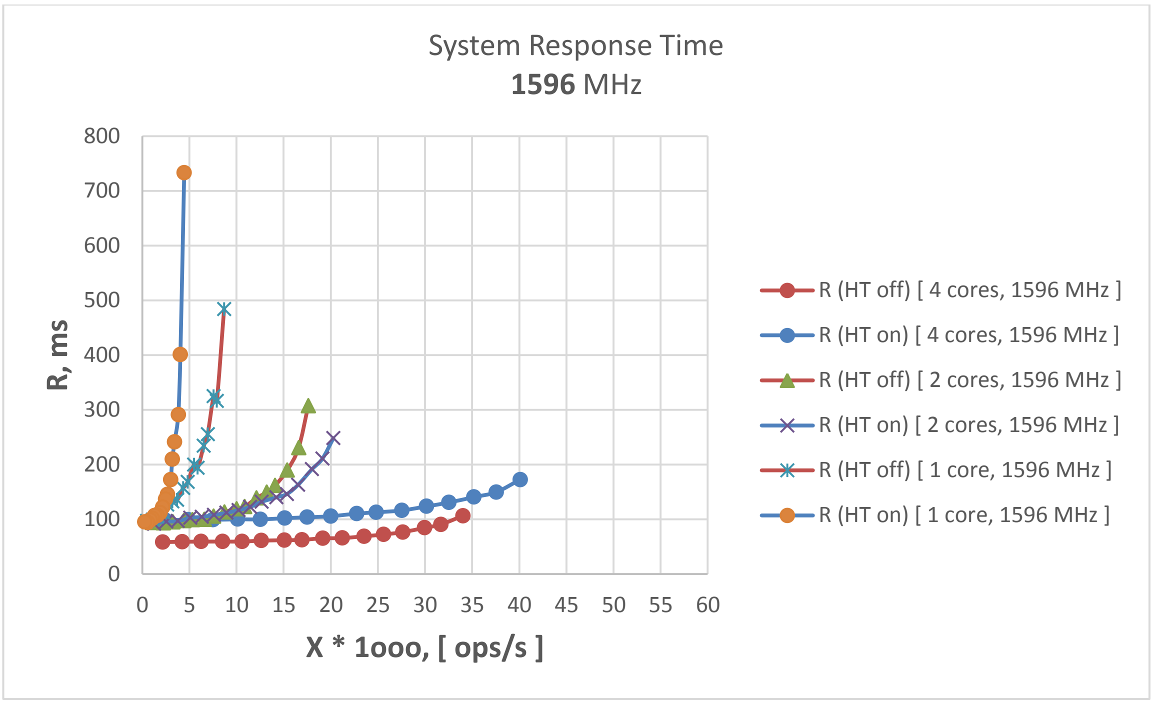

Figure 4.16: System Response Time with HT {on, off}
for {4, 2, 1} cores at 1596 MHz

It can be observed that with active HTT significant response time decrease is achieved at "weaker" configurations having low number of cores – one or two, while, absolutely unexpectedly, some performance degradation is observed for configurations with max number of cores. So, at 1596 MHz it turns out that the system without HTT would outstrip corresponding system with HTT for any workload level.

4.4.4 Conclusion

In this chapter we performed experimental validation of two-way simultaneous multithreading, or equivalently, Intel's Hyper-Threading technology. The following conclusions can be made from the obtained results:

- Performance of HTT is strictly dependent on hardware configuration and on type of software used.
- Performance growth may achieve from 9 to 33 % depending on the above factors and application scenario, which means that at some moments with HTT the capacity of the system, can be raised by one third over total system capacity. This very important results and it is the main reason why underestimation of the technology may result to enormous capacity wastes on a datacenter scale if neglect the effect of HTT. It becomes absolutely clear that Utilization law, which establishes linear relationship between utilization and throughput in computing systems and serves as the basis for capacity planning in large data centers is not suitable anymore in the presence of SMT, since it does not account effects enabled by SMT. For this reason, new "adequate" models accounting for SMT effects need to be developed. In next chapter we propose two such models, validation of which will be performed later in Chapter 6.
- As experiments reveal, much greater technology impact can be expected at "weaker" configurations having ones or two cores active. Most likely, relatively low performance increase at maximum configurations explained by the fact that we get high number of request simultaneously competing for limited shared and partitioned between them processor resources, like cache memories, load and store buffers. Another possible reason that may help to explain this fact is workload parallelization constraint of benchmarks, that is to say, a workload does not scale well with number of threads. If this is the case, taking into account also the fact the benchmarks offer vary intensive workload, able to utilize all system resource, we can very roughly set "effective threads threshold" for modern computing systems that may slightly vary from four to eight, while subsequent increase will apparently add only no to negligible performance increase.
- Another discovered positive aspect of technology is impact on system response time

impacting user waiting time. HTT allows significantly decrease user waiting time, from 15% to 20% depending application scenario. Surprisingly, there also have been revealed some cases when HTT slightly decrease this performance characteristic – at system configurations with max number of cores.

- All obtained results justify Intel’s claims about usage advantage of Hyper – Threading Technology.

Chapter 5

Proposed models

In current chapter we propose two models of SMT/HTT processor, developed using birth-death Markov chain model and Queuing Network model with Finite Capacity Region (FCR).

5.1 Introduction

In the real world, the main part of the work is about building the right model, that correctly characterizes the system and estimating the parameters of such model.

The success of queuing network modeling is based on the fact that the low-level details of a system are largely irrelevant to its high-level performance characteristics [17]. Queuing Network (QN) models appear abstract when compared with other approaches to computer system analysis. The underlying philosophy is to begin by identifying the principal components of the system and the ways in which they interact, and then supply any details that prove to be necessary [15][17].

Modeling cycle begins with the definition of the model, which includes selection of those system resources and workload components that will be represented, identification of any system characteristics that may require special attention (e.g., priority scheduling, paging), choice of model structure (e.g., separable, hybrid), and procedures for obtaining the necessary parameters from the available measurement data.

This feature of QN modeling permitted us to concentrate our attention directly on modeling of the principal component of this research – processor, and omitting direct representation of details related to other components of the system, like disks or memory.

Next, the system is gauged to obtain *workload measures*, from which model inputs will be calculated, and *performance measures*, which will be compared to model outputs. In some cases these are the same; for instance, device utilizations are workload measures, since they are used to calculate service demands, and also performance measures, since they are used to assess the accuracy of the model.

The workload measures then are used to parameterize the model, a step that may require various transformations. After, the model is evaluated, yielding outputs. These are compared to the system's outputs. Discrepancies may indicate flaws in the process, such as system characteristics that were ignored or represented inappropriately, or model inputs whose values were established incorrectly. An overview of the whole modeling process is illustrated on Figure 5.1.

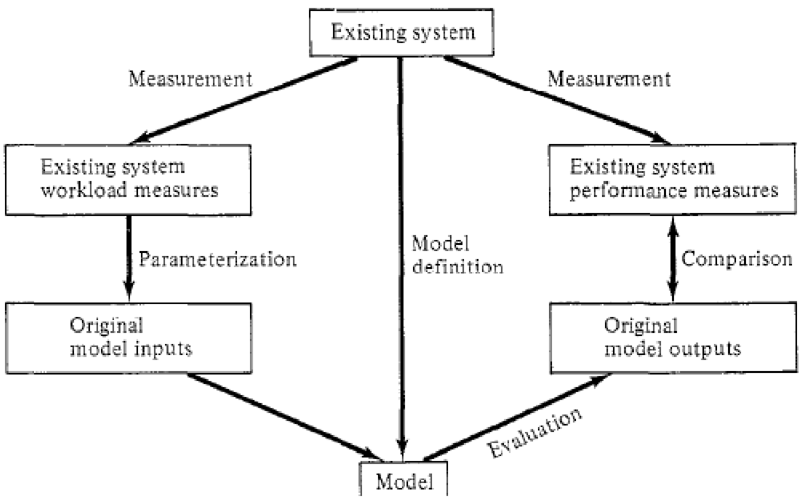

Figure 5.1: Modeling cycle overview.

5.2 SMT/HTT Model 1: Birth-Death μ/α Markov - Chain model

5.2.1 Introduction

As it was mentioned, in our work we restrict our attention on Intel's Hyper-Threading Technology, which, in fact, represents two-way Simultaneous Multithreading – case when each physical present core is recognized by the operating system as two virtual ones.

For the sake of simplicity, we consider exactly two-way simultaneous multithreading (SMT), which can be found on recent Intel processors based Nehalem, Westmere, Sandy Bridge and Atom micro-architectures. The model, however, can be easily extended to four – way SMT, adopted in the Intel's Xeon and IBM Power7 processors, and eight – way SMT, implemented in the Oracle Sparc T3 processors.

The proposed model based on two parameters (μ and α) and can be considered as a special case of the birth-death Markov model presented in Chapter 3.

5.2.2 Model definition

The key idea underlying is that SMT processor is best modeled as a load-dependent service center, while superscalar processor can be easily modeled as simple service queue and, secondly, SMT processor exhibits linear growth up with number of cores; however after activation of SMT the growth is not linear anymore and represented by parameter α.

Let m to be the number of CPU cores, and μ to be the service rate when a single job is in the system. Under the assumption that the operating system scheduler is aware of SMT, when $k \leq m$ jobs are present in the system, they are placed on the separate CPU cores. In this case, assuming low contention for memory and shared caches, the service rate of the system becomes $m\mu$.

SMT allows us to exploit up to m additional jobs. When $m < k \leq 2m$, threads are scheduled, the service rate of the system is increased, since $2m - k$ cores are still completing requests at rate μ, while the other $k - m$ cores are completing requests at rate $\mu(1 + \alpha)$ due to increased utilization of computational units [22]. Therefore, the total processing rate is equal to:

$$X_{HTon,max} = m\mu + (k - m)\alpha\mu \qquad (5.1)$$

The parameter α is heavily workload dependent and its estimation is discussed in the following section.

For $k \geq 2m$, the overall service rate is $m(1 + \alpha)\mu$ and equals to the maximum sustained throughput of the system.

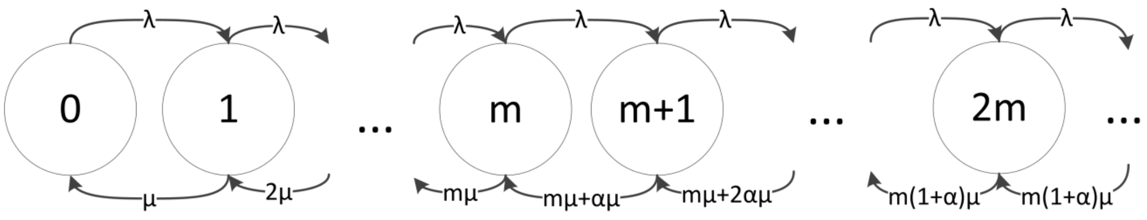

Figure 5.2: The Markov Chain representing a system with m CPU cores and two – way SMT.

The steady state probabilities p_k for $k \geq 1$ are:

$$p_k = p_0 \prod_{i=0}^{k-1} \frac{\lambda}{(i+1)\mu} \qquad 1 \leq k \leq m \qquad (5.2)$$

$$p_k = p_0 \prod_{i=0}^{k-1} \frac{\lambda}{(i+1)\mu} \prod_{i=m}^{k-1} \frac{\lambda}{(i+1-m)\alpha\mu} \qquad m < k \leq 2m \qquad (5.3)$$

$$p_k = p_0 \prod_{i=0}^{k-1} \frac{\lambda}{(i+1)\mu} \prod_{i=m}^{k-1} \frac{\lambda}{(i+1-m)\alpha\mu} \prod_{i=2m}^{k-1} \frac{\lambda}{m(1+\lambda)\mu} \qquad 2m < k \qquad (5.4)$$

Defining the load intensity $\rho = \frac{\lambda}{m(i+1)\alpha\mu}$, these equations can be rewritten as follows:

$$p_k = p_0 \prod_{i=0}^{k-1} \frac{(m(1+\alpha))^k \rho^k}{k!} \qquad 1 \leq k \leq m \qquad (5.5)$$

$$p_k = p_0 \frac{(m(1+\alpha))^k \rho^k}{m! \prod_{i=m}^{k-1}(m+(i+1-m)\alpha)} \qquad m < k \leq 2m \qquad (5.6)$$

$$p_k = p_0 \frac{(m(1+\alpha))^{2m} \rho^k}{m! \prod_{i=m}^{2m-1}(m+(i+1-m)\alpha)} \qquad 2m < k \qquad (5.7)$$

By definition of probability:

$$\sum_{k=0}^{inf.} p_k = 1 \qquad (5.8)$$

It follows that the steady state probability of an empty queue is:

(5.9)

$$p_0 = \left(1 + \sum_{k=1}^{m} \frac{\left(m(1+\alpha)\right)^k \rho^k}{k!} + \sum_{k=m+1}^{2m} \frac{\left(m(1+\alpha)\right)^k \rho^k}{m! \prod_{i=m}^{k-1}(m+(i+1-m)\alpha)} \right.$$
$$\left. + \frac{\left(m(1+\alpha)\right)^{2m}}{m! \prod_{i=m}^{2m-1}(m+(i+1-m)\alpha)} \sum_{k=2m+1}^{inf.} \rho^k \right)^{-1}$$

Writing the rightmost summation as $\rho^{2m+1}\sum_{k=0}^{inf.}\rho^k$ and observing that $\rho < 1$ is required for stability, we can use the proof of convergence for geometric series and write p$_0$ as follows:

(5.10)

$$p_0 = \left(1 + \sum_{k=1}^{m}\frac{\big(m(1+\alpha)\big)^k\rho^k}{k!} + \sum_{k=m+1}^{2m}\frac{\big(m(1+\alpha)\big)^k\rho^k}{m!\prod_{i=m}^{k-1}(m+(i+1-m)\alpha)} + \frac{\big(m(1+\alpha)\big)^{2m}\rho^{2m+1}}{m!\prod_{i=m}^{2m-1}(m+(i+1-m)\alpha)}\frac{1}{1-\rho}\right)^{-1}$$

The average number of jobs in the queue is:

(5.11)

$$E[N] = p_0\left(\sum_{k=1}^{m}\frac{\big(m(1+\alpha)\big)^k\rho^k}{k!}k + \sum_{k=m+1}^{2m}\frac{\big(m(1+\alpha)\big)^k\rho^k}{m!\prod_{i=m}^{k-1}(m+(i+1-m)\alpha)}k + \frac{\big(m(1+\alpha)\big)^{2m}}{m!\prod_{i=m}^{2m-1}(m+(i+1-m)\alpha)}\sum_{k=2m+1}^{inf.}\rho^k\,k\right)^{-1}$$

The infinite series can be simplified with some simple algebraic operations and using convergence of the geometric series:

$$\sum_{k=2m+1}^{inf.}\rho^k\,k = \rho\sum_{k=2m+1}^{inf.}\rho^{k-1}\,k \qquad (5.12)$$

$$= \rho\sum_{k=2m+1}^{inf.}\frac{d}{d\rho}\rho^k \qquad (5.13)$$

$$= \rho\frac{d}{d\rho}\rho^{2m+1}\sum_{k=2m+1}^{inf.}\rho^k \qquad (5.14)$$

$$= \qquad \rho \frac{d}{d\rho} \rho^{2m+1} \frac{1}{1-\rho} \tag{5.15}$$

$$= \qquad \rho \left((2m+1)\rho^{2m} \frac{1}{1-\rho} + \rho^{2m+1} \frac{1}{(1-\rho)^2} \right) \tag{5.16}$$

$$= \qquad \frac{\rho^{2m+1}}{1-\rho} \left((2m+1) + \frac{\rho}{1-\rho} \right) \tag{5.17}$$

From Little's law [17] we can find Response time:

$$R = \frac{E[N]}{X} \tag{5.18}$$

Since each job is placed on separate thread, utilization is defined as number of "busy" threads:

$$U = \sum_{k=1}^{2m} Pk * \frac{k}{2m} + \sum_{k=2m+1} Pk * 1 \tag{5.18}$$

5.2.3 Alpha (α) estimation: standard way

In the context of the model, parameter Alpha indirectly defines performance gain enabled by HT technology activation and it is heavily workload dependent. There have been defined two ways in which parameter α can be estimated.

First way is a standard way, in which α is calculated based on maximum throughputs obtained with HT on and HT off:

$$X_{HTon,max} = \mathrm{m}(1 + \alpha)\mu \tag{5.19}$$

$$X_{HToff,max} = \mathrm{m}\mu \tag{5.20}$$

$$\alpha = \frac{X_{HTon,max}}{m\mu} - 1 \tag{5.21}$$

$$\alpha = \frac{X_{HTon,max}}{X_{HToff,max}} - 1 \tag{5.22}$$

Where, m – number of physical cores; μ – service rate when a single job is in the system. $X_{HTon,max}$ is a maximum system throughput measured with HT being activated and $X_{HToff,max}$ is a maximum system throughput measured without HT technology.

An advantage of this technique is relative ease of computation, since we can supply the parameters to our model by performing only two experiments.

5.2.4 Alpha (α) estimation through error minimization

The optimization technique has been developed to further increase accuracy of the defined above model, and in fact can be applied to optimize any analytical models. Optimization application does make sense only in the presence of noticeable discrepancies between system and model data. As a rule, it is applied after first model evaluation phase to reduce discrepancies between such system and model performance measures, like *system response time* and *processor utilization*. The process itself consists in finding of optimal value of parameter Alpha (α), by that enabling maximum fitting between selected system and model performance measures and reducing estimation error.

5.2.4.1 Mean Squared Error (MSE) estimation

Optimization is based on estimation and following minimization of Mean Squared Error (MSE). MSE of a model is one of many ways to quantify the difference between values implied by the model and the true values of the quantity being estimated. MSE measures the average of the squares of the "errors". The error is the amount by which the value implied by the model differs from the quantity to be estimated. Why such “errors” occur is mentioned in introduction of the chapter.

The MSE is the second moment (about the origin) of the error, and thus incorporates both the variance of the model and its bias. Like the variance, MSE has the same units of measurement as the square of the quantity being estimated.

If $\hat{Y}$ is a vector of n model estimations, and Y is the vector of true values, obtained after system measurement, then the estimated MSE of the model is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{5.23}$$

Since, the number of system measurements equal n depends on the type of benchmark, then, for SPEC Power SSJ 2008 benchmark MSE becomes equal to:

$$MSE = \frac{1}{10}\sum_{i=1}^{10}(\hat{Y}_i - Y_i)^2 \tag{5.24}$$

And for Open-load benchmark MSE is:

$$MSE = \frac{1}{16}\sum_{i=1}^{16}(\hat{Y}_i - Y_i)^2 \tag{5.25}$$

For minimization of MSE a GRG Nonlinear and Evolutionary Engine solving method have been exploited, supplied with Excel Develop tools.

5.2.5 Model Validation

Validation of this and other presented in this work models, along with aposterior optimization, is performed in next chapter.

5.3 SMT/HT Model 2: Queuing Network Model with Finite Capacity Region (FCR)

5.3.1 Introduction

Among existing modeling techniques, Queuing Networks with Finite Capacity Regions (FCR) have largely proven to be effective in characterizing simultaneous resource possession in which a customers or requests can hold more resources simultaneously. Such queuing networks impose upper bounds on the number of jobs that can simultaneously reside in a set of stations [21][25], and can be used to model also high-level application constraints. These are sub networks where the number of circulating jobs is constrained.

Finite capacity regions are characterized by two types of constraints: a dedicated constraint bounds the number of jobs in a region for a specified class; a shared constraint limits the number of jobs without class distinctions. In our model we impose only shared constraint since all jobs belong to only one class [21].

Jobs arriving to a Finite Capacity Region enqueue in a waiting buffer outside the region. The presence of the waiting buffer makes it difficult to obtain an analytical solution to models with Finite Capacity Regions. Therefore, only approximation techniques have been developed. However, simulation remains the most important analysis technique for models with FCR in presence of realistic workloads.

5.3.2 Model definition

The key underlying idea in the model is that at any instant of time only two requests, and no more, can be simultaneously processed in two-way SMT processor, or equivalently, in Intel's HTT processor.

Single core two-way SMT processor model is illustrated on the Figure 5.3. Area surrounded by bold shaded line represents Finite Capacity Region with simultaneous possession capacity equal to two jobs at any instant of time.

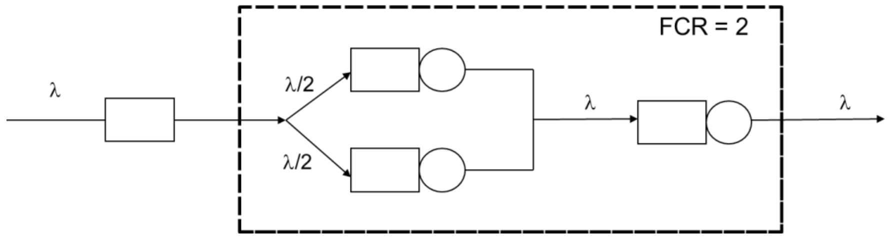

Figure 5.3: Queuing Network Model with FCR for single core two-way SMT processor.

In some sense the model can be considered as a possible enhancement of the model proposed by authors from BMC software and described in corresponding section of Chapter 3. Similarly, duplicated parts of the processor, which enable support of extra architectural state, are represented in parallel subpart of FCR, where two identical service centers are arranged in parallel. The third service center (sequential part of FCR) represents all resources that competitively shared by requests in FCR or completely unaware of SMT resources, like execution units.

Again, from the application point of view each job visits one of the two logical processors offered to it by the operating system. On the chip, each instruction execution still goes through two phases. On the first one, an instance of the duplicated part of the hardware does preparatory work in parallel part of FCR, looking up data and instructions in the cache or getting them from memory and then in sequential part instruction is executed.

Jobs arriving to FCR are queuing in a buffer outside the region, if the FCR limits are exceeded. The buffer is assumed to have infinite capacity to store “waiting” requests.

The situation can be modeled as following: when first request arrives, it first passes through instruction preparation phase and then gets directly executed, meanwhile, if there is a second request arriving, it occupies second “preparation” service center. In such a case there is no anymore place for any other requests, until first instruction receives service, all arriving request will get buffered outside FCR.

5.3.3 Model extension for dual-core processors

The model is obtained simply by scaling above model with respect to a number of cores, which in this case is equal to two. Figure 5.4 presents an extension for dual core processor supporting two-way SMT.

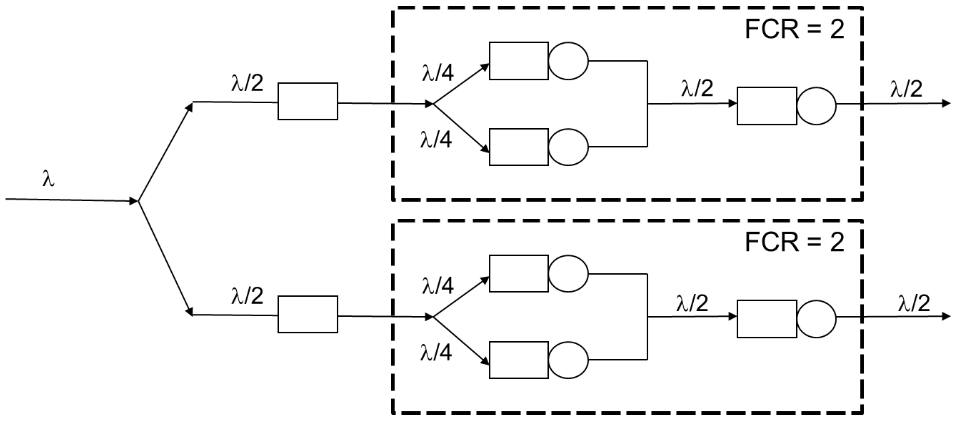

Figure 5.4: Queuing Network Model with FCR for single core two-way SMT processor.

Peculiarity of the model is that it has a separate waiting buffer for each physical core. From the application point of view, now each job can visit one of the four logical cores offered to it by the operating system (with active HTT). Since we assume that OS is aware of SMT presence, load balancing should work correctly distributing workload equally between physical cores.

As in previous case, within FCR, each instruction execution goes through two phases. First, an instance of the duplicated part of the hardware inside one of two cores does preparatory work, looking up data and instructions in the cache or getting them from memory; and then the single real CPU executes the instruction within the core.

5.3.4 Model extension for quad-core processors

Figure 5.5 illustrates a model of quad-core processor with HTT. The model is obtained by duplication of the processor core four times.

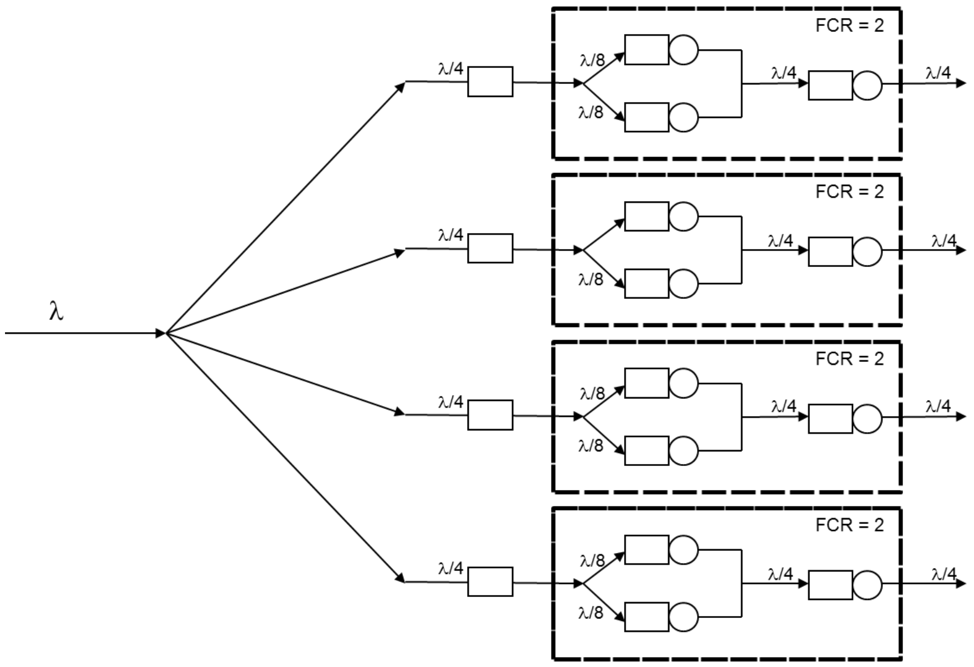

Figure 5.5: Queuing Network Model with FCR for single core two-way SMT processor.

The model characteristics remain the same as in case with Extension 1, but now OS recognizes eight virtual cores when HTT is active. Again assuming that operating system has correct load balancing mechanism, each virtual core would receive only forth part of the overall workload.

5.2.6 Model Validation

Because of non-availability of adequate analytical solution techniques for open QN models with FCR, the main technique used to solve such models is simulation. The principal strength of simulation is its flexibility. In this thesis the JMT (Java Modeling Tool) tool has been used to model and simulate the SMT/HTT queuing network model with Finite Capacity Region. All validation results are presented in next chapter.

Chapter 6

Validation and Comparison

This chapter pursues a double goal: to present validation results for SMT/HTT models that we developed in Chapter 5 and perform comparative analysis with other state of art QN models to identify models with lowest prediction error.

6.1 Test bed configuration

The system under test used for benchmarking to produce workload and performance measures has the same configuration as the one described in Chapter 4. For details please refer to "Test bed configuration" section of Chapter 4.

6.2 Methodology

After model has been defined, next step would be to measure the system under different workload levels for each examined system configuration, to obtain *workload measures,* which can be used directly to feed and parameterize our models, and *performance measures,* which will be compared to the model outputs.

Models are constructed for the following system configurations:

- 4 cores, HT enabled, 2660 MHz
- 4 cores, HT enabled, 1596 MHz
- 2 cores, HT enabled, 2660 MHz
- 2 cores, HT enabled, 1596 MHz
- 1 core, HT enabled, 2660 MHz
- 1 core, HT enabled, 1596 MHz

To generate workload and record performance measures two benchmarking suites are used: synthetic and SPEC Power SSJ 2008 benchmarks. More details about both of them can be found in dedicated subsections of Chapter 4. All validation data produced by benchmarks is also available in Chapter 4.

As we mentioned in Chapter 5, there are two alternative approaches to solve models: based on analytical modeling and based on computer simulation.

Analytical modeling is conducted by a mathematical analysis that characterizes a network as a set of equations, while simulation permits modeling of complex computing systems using special software tools. Simulation technique is appropriate tool for representing complex events, when analytical modeling becomes too complicated or when solution techniques, either exact or approximate, have not been yet proposed or are not precise.

In our work computer simulation is used to validate QN model with FCR proposed in previous chapter. The presence of the waiting buffer makes queuing network non-separable and makes difficult to obtain analytic solution especially for open models with FCR. Therefore, only approximation techniques have been developed. However, simulation remains the most important analysis technique in presence of realistic workloads.

All other presented in this work models are solved based on their analytical solutions.

6.2.1 Simulation

The principal strength of simulation is its flexibility. There are few restrictions on the behavior that can be simulated, so a computer system can be represented at an arbitrary level of detail. At the abstract end of this spectrum is the use of simulation to evaluate networks of queues. While the principal weakness of simulation modeling is cost of evaluation, because running a simulation requires substantial computational resources, especially if narrow confidence intervals are desired.

6.2.1.1 JMT tool

In this thesis the JMT (Java Modeling Tool) [28] tool has been used to model and simulate the SMT/HTT queuing network model with Finite Capacity Region.

JMT is a suite of applications developed by Politecnico di Milano and released under GPL license. The project of JMT offers a complete framework for performance evaluation, system tuning, and capacity planning and workload characterization studies.

One of the embedded JMT tools is JSIMgraph – a simulation application within the suite of JMT allows the design of a queuing network model in graphical way [25][27]. The model can be solved either with simulation techniques or, if the model is BCMP compliant, automatically exported to JMVA to be solved with exact or approximate Mean Value Analysis (MVA) algorithm.

6.2.1.2 JSIM model simulation

As it mentioned in [27] JSIM simulation module of the Java Modeling Tools (JMT) is an open-source fully-portable Java suite for capacity planning studies. The simulator has been purposely developed to help both inexperienced and advanced users. Most of the difficult decisions that are needed in order to run simulations properly, such as the detection of the transient part of samples to be discarded, have been automated. The tool also provides guidance over the graphical design of the network and over the analysis and the plot of the results. What-if parametric analysis for parametric evaluation of complex systems is supported. Several features that increase the generality of the applications to capacity planning studies are provided, among them fork-join service centers, regions with finite capacity, state-dependent routing algorithms, priority classes and import of real workload distributions from log files. The JSIMgraph has been designed to be very flexible. The important feature is separation between the GUI and computation engine by introducing an XML layer as shown in Figure 6.2.

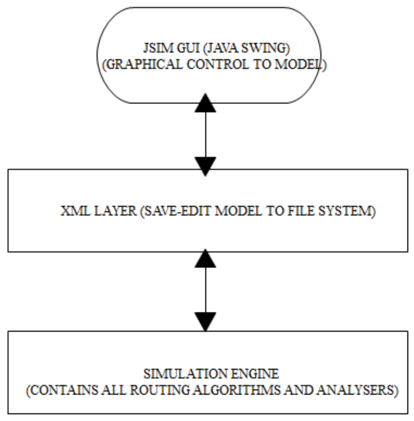
JSIM GUI (JAVA SWING)
(GRAPHICAL CONTROL TO MODEL)
XML LAYER (SAVE-EDIT MODEL TO FILE SYSTEM)
SIMULATION ENGINE
(CONTAINS ALL ROUTING ALGORITHMS AND ANALYSERS)

Figure 6.2: JSIM Framework

The tool can be divided in 3 layers.

1. GUI layer (Graphical User Interface to design and configuring models)
2. XML layer (saving and reading models to file system)
3. SIMULATION layer (actual implemented strategies and logic)

In JSIMgraph, the model of Queuing Network created using the Graphical User Interface (i.e. GUI) is saved in the form of a XML. The graphical user interface developed using Java Swing Architecture. The GUI modifies the model which is saved in the java class, which is then converted into an XML file using the JSIM.xsd file. The architecture allows reuse of the simulation or computation engine by other tools or projects by providing a suitable XML input file. At the end of the computation and simulation the performance measures and indices are in to the solution element of input file.

The core module of the simulation engine is a discrete event calendar that acts as messaging service provider, sending messages to simulation engines. The arrival of a job to the queuing station and corresponding departure after service completion is represented by message. When all the events for a given time are processed, the simulation current time is moved forward to the next instant with a given time in the calendar.

In the simulation network, each service station is composed of three parts, called Sections. The three sections are named as Input section, Service section and Output section. The service demands are specified through the GUI interface in the service section of the queuing station. In a queuing station the input section receives incoming jobs to be processed by the service section. The service section simulates the service process with the user specified service time distribution. After service completion, jobs are forwarded to the output section called Router which sends the request to the input section of another queuing or service center according to user specified routing policy.

6.2.1.3 Parameters estimation

The service demands for service centers inside FCR for QN models illustrated on Figures 6.3 – 6.5 have been estimated with trial and error method. Since model, in some sense, enhances the model of authors from BMC software presented in Chapter 3, the optimal parameter values of their model have been taken as a starting point (estimation is discussed in corresponding section of Chapter 3). Obviously, service demands values inside FCR have to be smaller to process the same amount of request, since only two requests can simultaneously reside inside region. By trials and errors, preserving service demand ratios, the parameters have been found and are listed in the Comparison section of the Chapter.

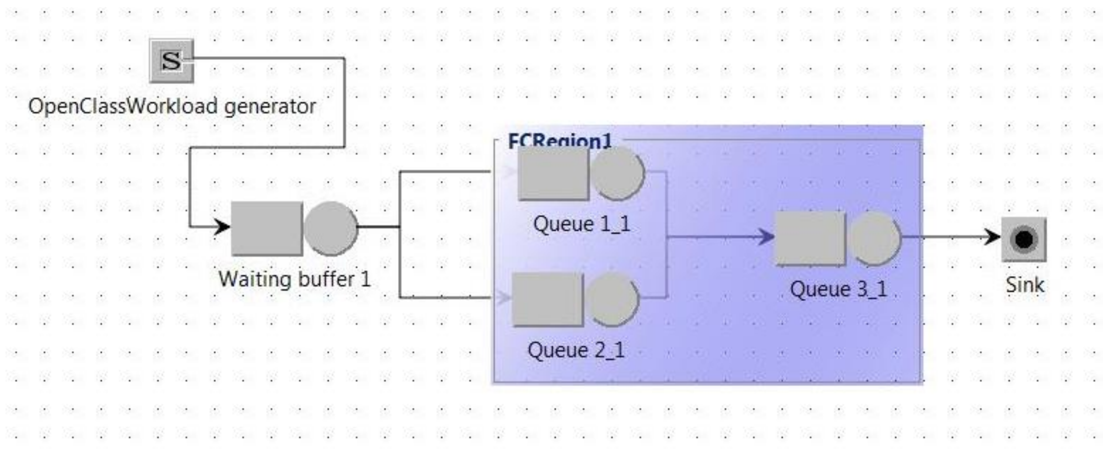

Figure 6.3: QN model with FCR for 2-way single-core SMT processor

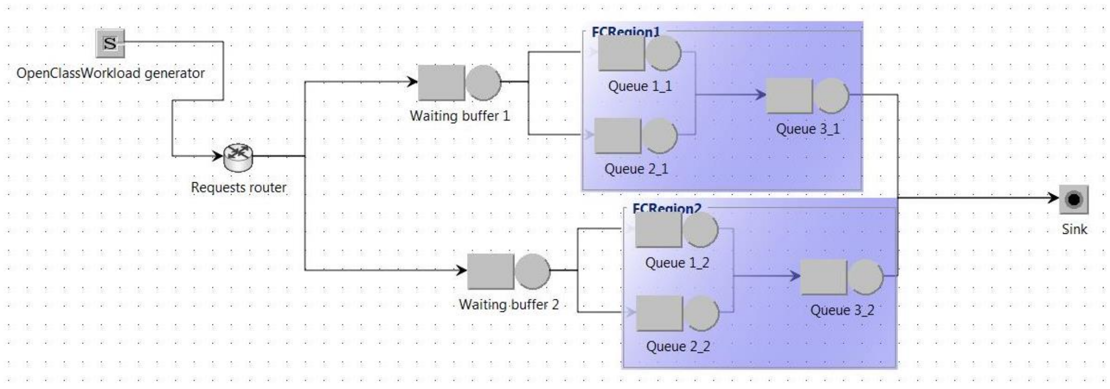

Figure 6.4: QN model with FCR for 2-way Dual-core SMT processor

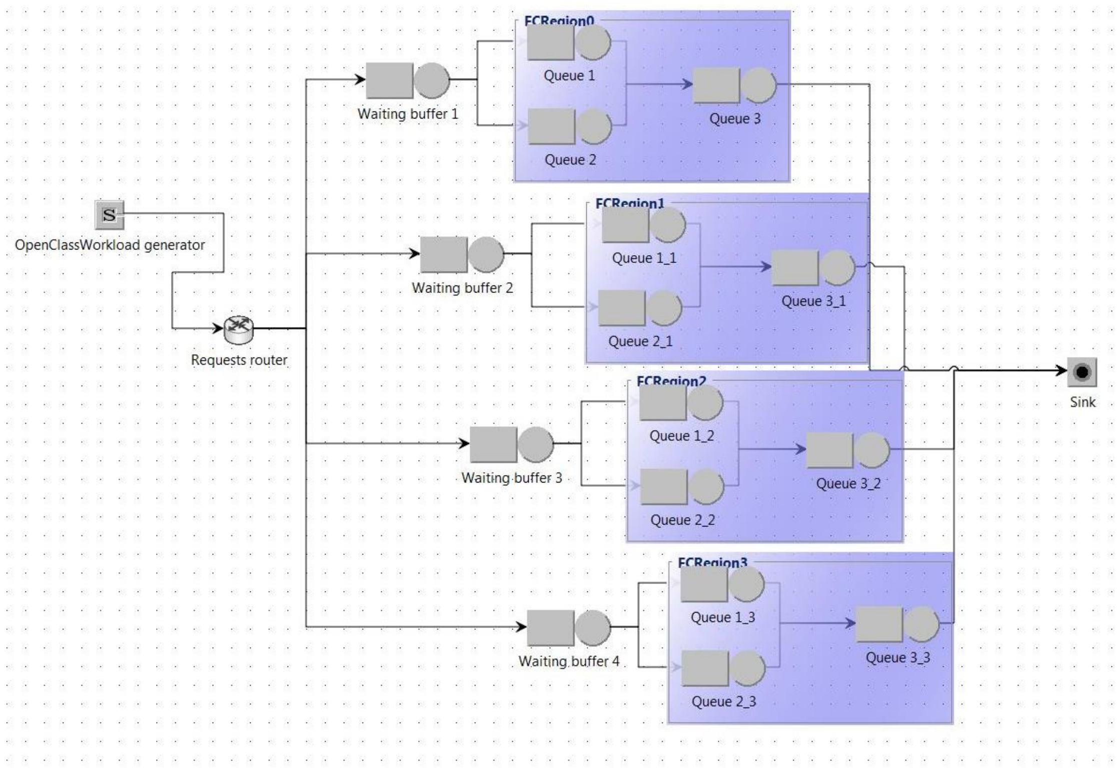

Figure 6.5: QN model with FCR for 2-way Quad-core SMT processor

6.3 Models Validation

The main performance characteristic for model accuracy estimation is *System Response Time* is also utilized as main measure for comparative analysis between models.

Utilization data from all benchmark runs is used to validate developed birth-death μ/α Markov chain model alongside with different parameter estimation techniques.

The series of plots listed below depict *System Utilization* versus *Model Utilization* for different system configuration for both benchmarks with parameters estimated in a standard way (details can be found in Chapter 5). Parameter values can be found in right upper corner of each plot.

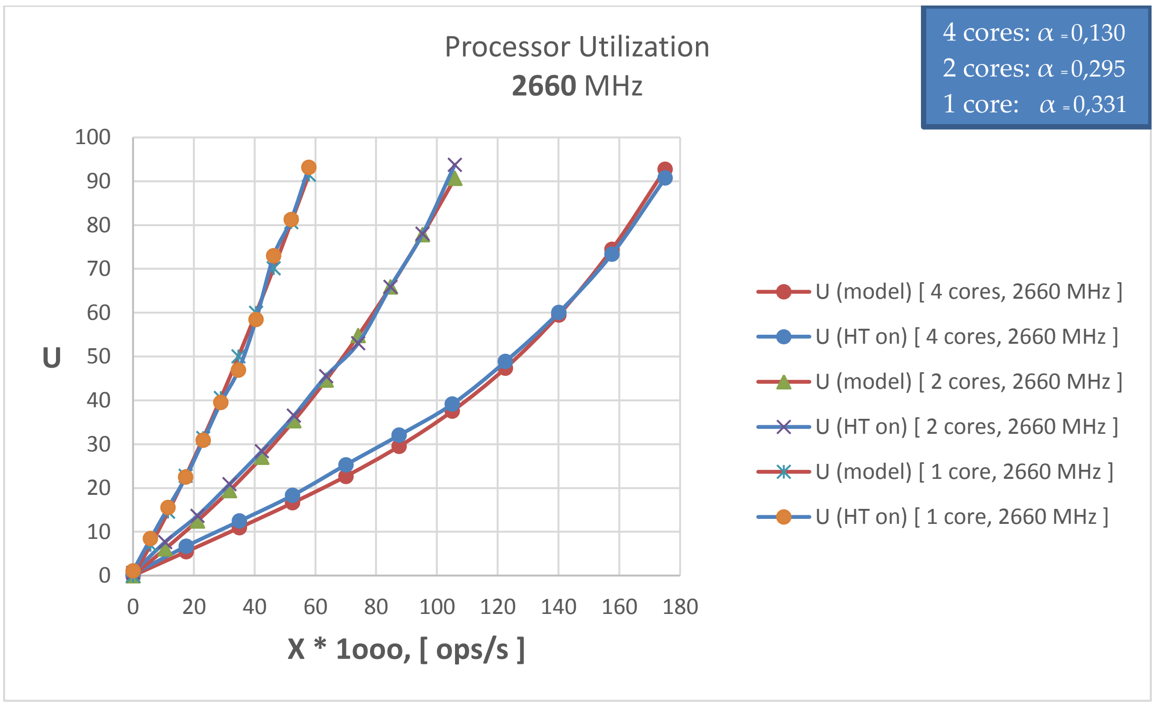

Figure 6.6: SPEC benchmark: {System vs. Model} Utilization with HT for {4, 2, 1} cores at 2660 MHz

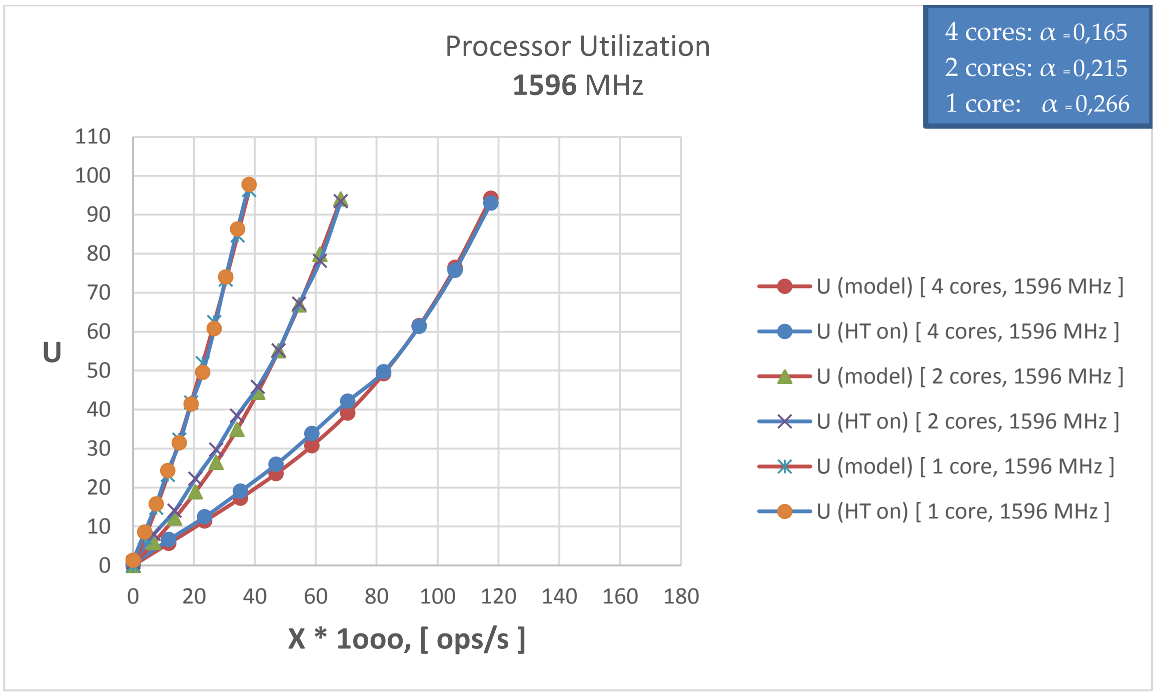

Figure 6.7: SPEC benchmark: {System vs. Model} Utilization with HT for {4, 2, 1} cores at 1596 MHz

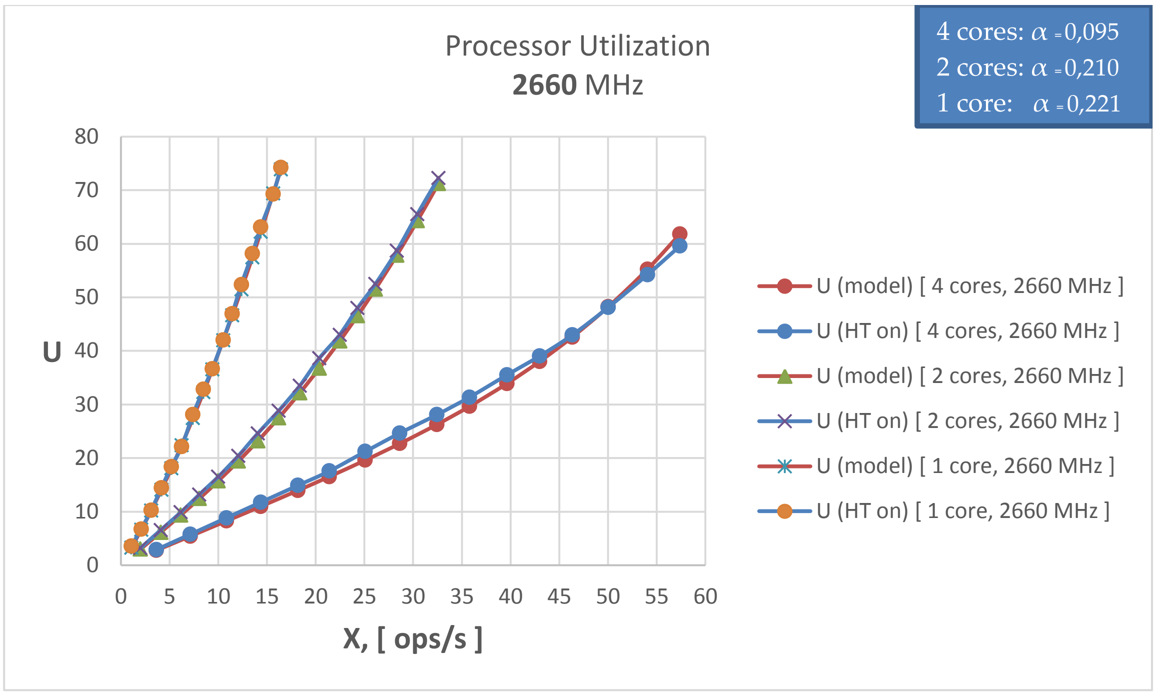

Figure 6.8: Synthetic benchmark: {System vs. Model} Utilization with HT for {4, 2, 1} cores at 2660 MHz

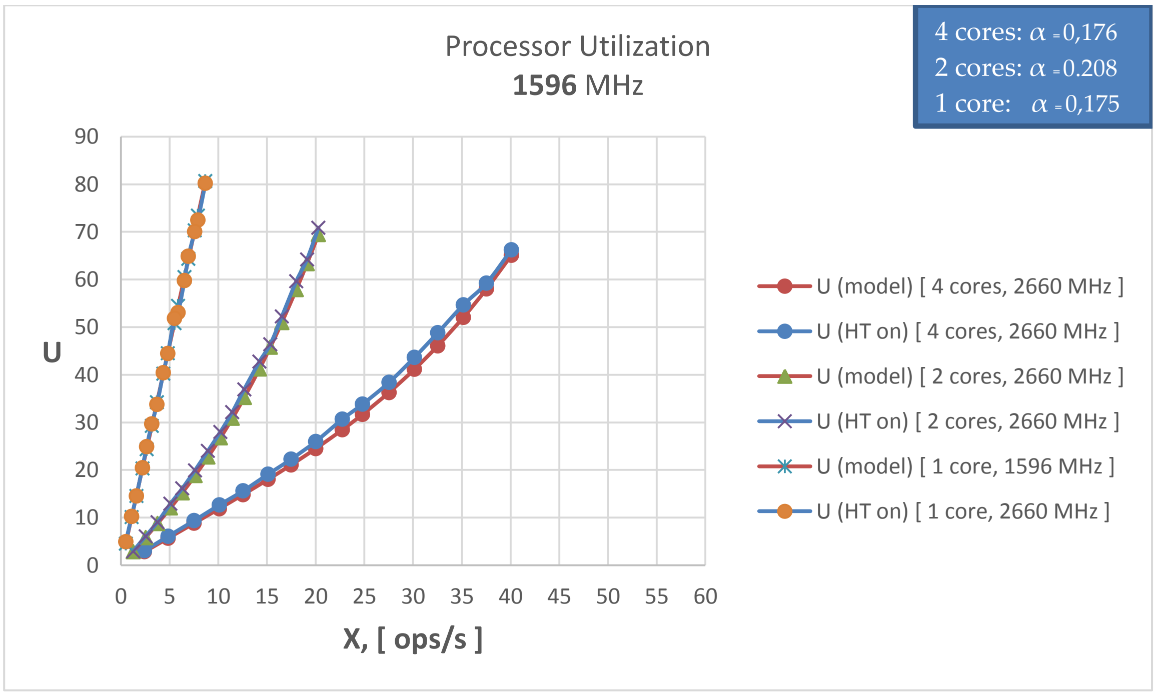

Figure 6.9: Synthetic benchmark: {System vs. Model} Utilization with HT for {4, 2, 1} cores at 1596 MHz

From the above figures we can say that birth-death μ/α Markov chain model seems to be very sound because of truly accurate results for utilization. Just some fairly negligible discrepancies that stay within experimental error are present for four cores configuration.
We can summarize that model possesses great predictive capacity for utilization measure.

6.3.1 Parameter estimation based on error minimization

Birth-death μ/α Markov chain model has been evaluated not only for *Utilization*, but also for *Response Time* performance measure, and then yielding outputs has been compared to the system's performance measures. For system Response time some configurations, primarily the ones with two and four cores, exhibited small discrepancies which stay within 5% to 10% error. It may indicate some flaws in the process or some small system characteristics that were ignored or represented inappropriately. However, achieve better model results and to further reduce error to 3-5%, parameter estimation based on error minimization technique can be applied. More in detail described in Chapter 5. In our case it is intended to reduce the model error for Response time and it is strictly dependent on the estimation of parameter Alpha.

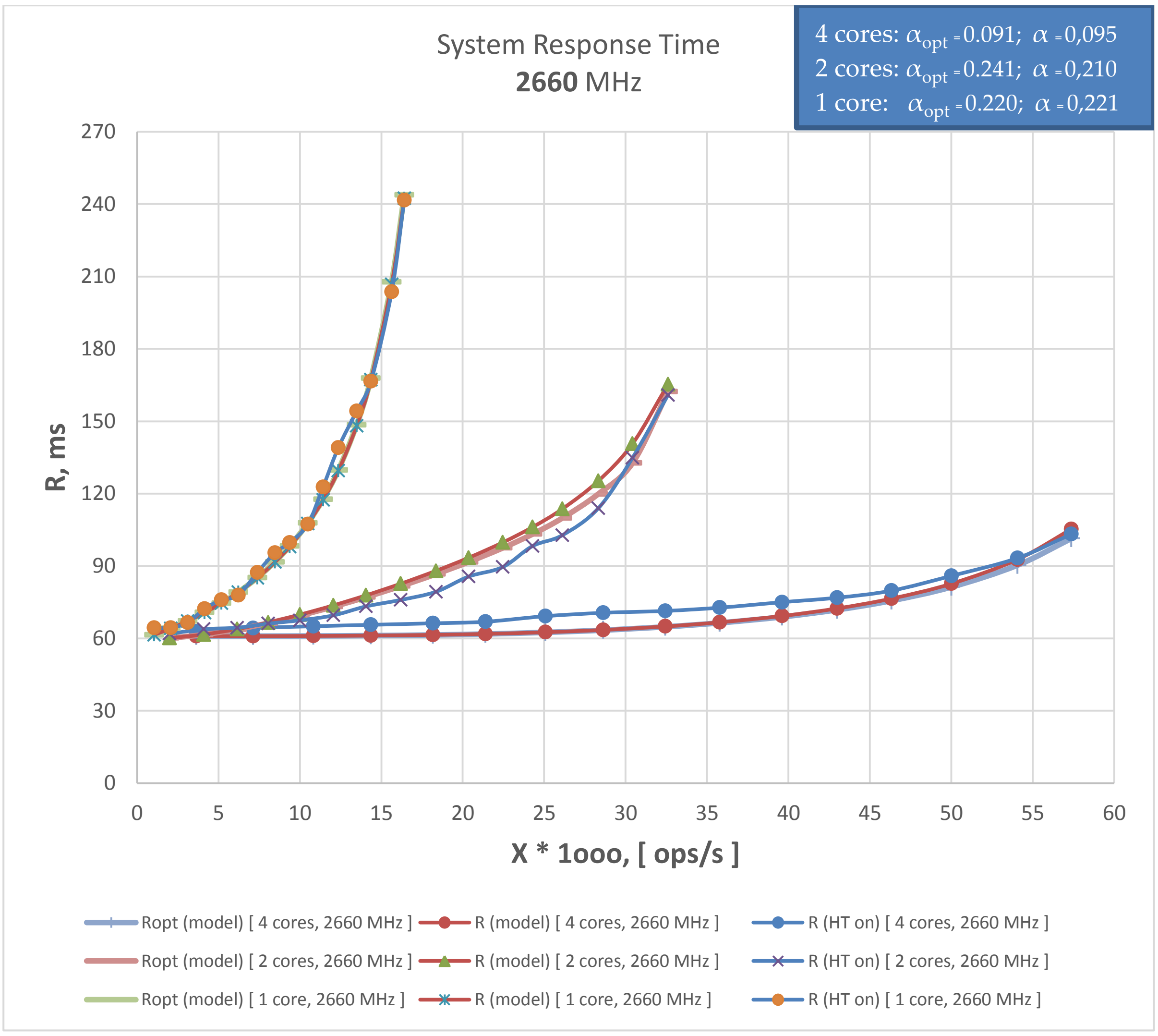

Figure 6.10: Synthetic benchmark: {System vs. Model}
Response time with HT for {4, 2, 1} cores at 2660 MHz

As we it can be noticed from the Figures 6.10 and 6.11 the values of parameter α are tuned so that to achieve the best possible fitting with original synthetic dataset. Especially it can be observed on Figure 6.11 representing scaled version of 6.10.

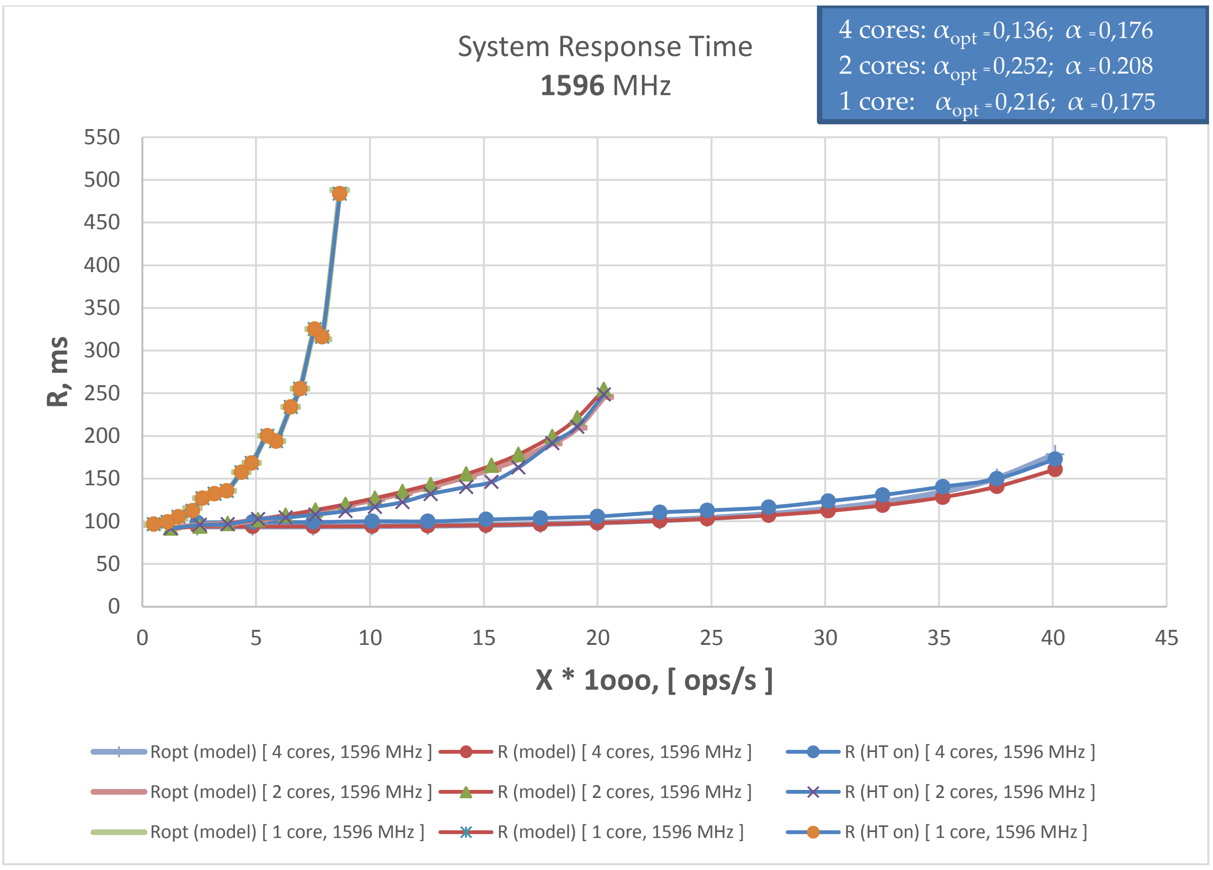

Figure 6.11: Synthetic benchmark: {System vs. Model}
Response time with HT for {4, 2, 1} cores at 1596 MHz

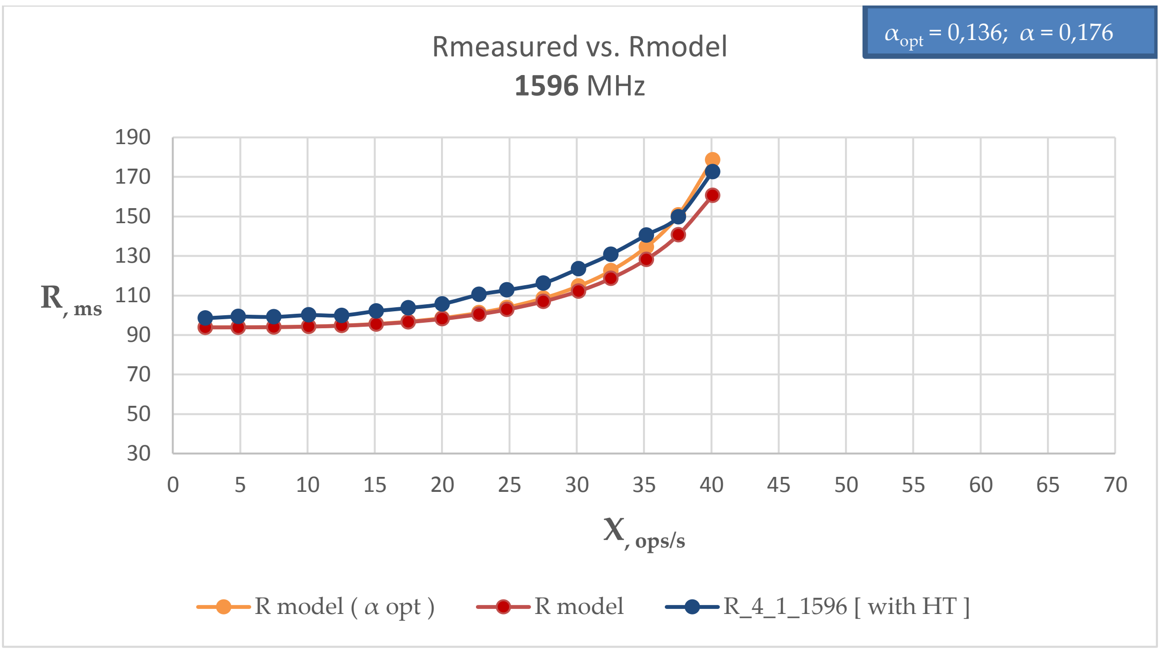

Figure 6.12: Response Time after optimization for 4 cores,
1596 MHz with HT

6.4 Comparison

Figures 6.13, 6.14 and 6.15 illustrate comparison results between models for synthetic benchmark data. The following models have been confronted:

- **M/M/1 Queuing Network model:** on plots referred as “M/M1”
- **M/M/2 Queuing Network model:** on plots referred as “M/M2”
- **M/M/4 Queuing Network model:** on plots referred as “M/M4”
- **SMT/HTT Model 1: Queuing Network model**: Two-way SMT QN model proposed by authors from BMC Software and described in detail in Chapter 3. On plots referred as “SMT/HTT QN model”.
- **Queuing Network model with Finite Capacity Region (FCR)**: This model is considered rather as enhancement of “SMT/HTT QN model”, which restricts the number of requests that can be served at any instant of time to two. Complete description of the model can be found in Chapter 5. On plots referred as “QN model with FCR”.
- **Birth-death $\boldsymbol{\alpha/\mu}$ Markov chain model**: Model possessing highest accuracy, analytical model of which can be found in Chapter 5. On plots referred as “Markov chain model”

From the Figures 6.13 – 6.15 it can be observed that the best predictive capability in all considered cases possess “Markov chain model” and “QN model with FCR”. They provide the best fitting with system measured data.

At the same time “M/M/x” models provide acceptable results, outperforming in some cases more sophisticated “SMT/HTT QN model”. “SMT/HTT QN model”, instead, improve results with increment of number of cores.

Below each plot, for the models the reader can find estimated relevant parameters.

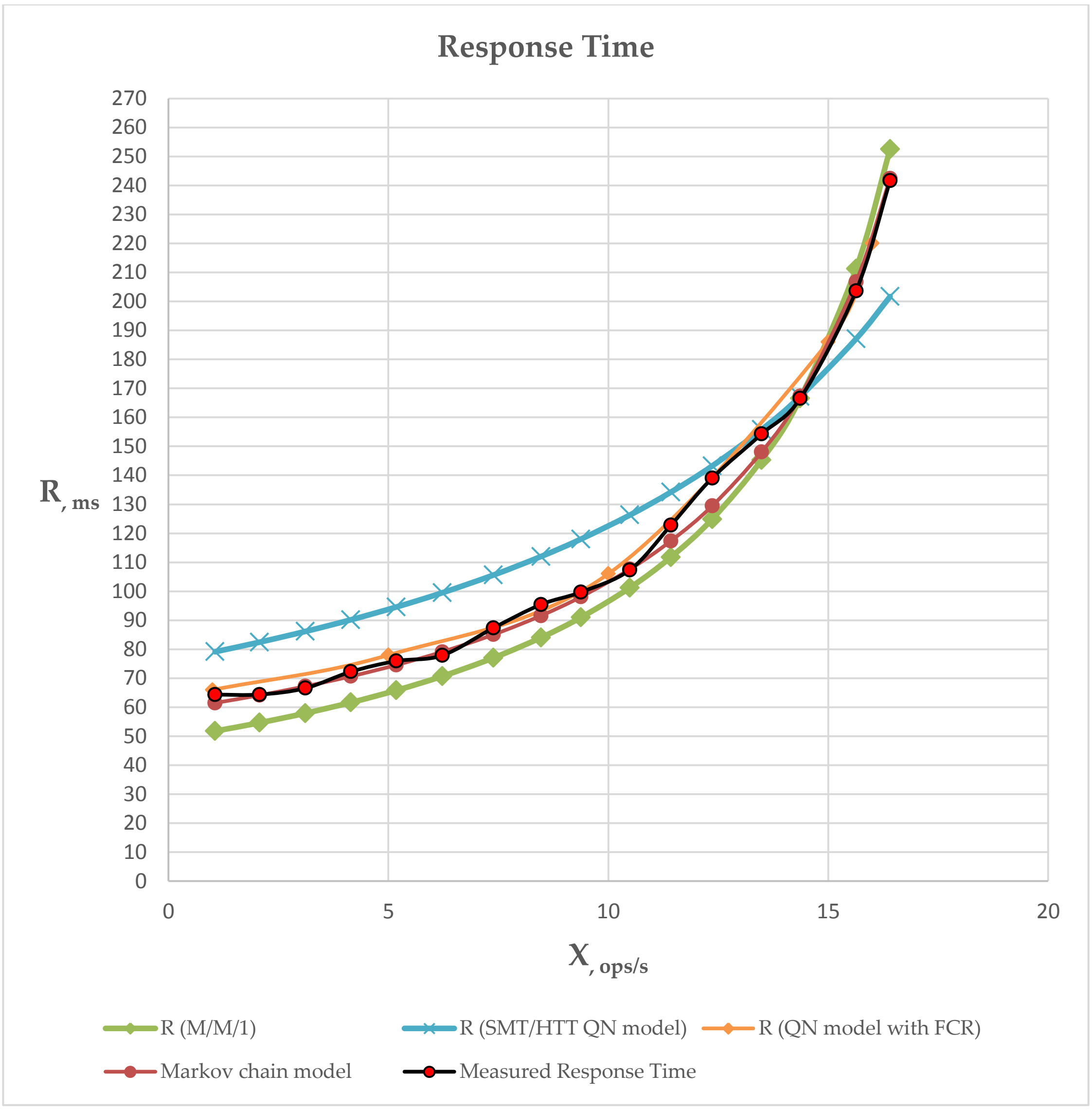

Figure 6.13: Response Time measure for 1 core, 2660 MHz with HT

On the figure 6.13:

- For “Markov chain model”: α has been estimated to be 0,221.
- For “M/M/1”: service time s has been estimated to be 0,049.
- For “SMT/HTT QN model”: $s1$ and $s2$ are equal to 0,076ms and 0,001ms respectively.
- For “QN model with FCR”: $s1$ and $s2$ are equal to 0,063ms and 0,0001ms respectively.

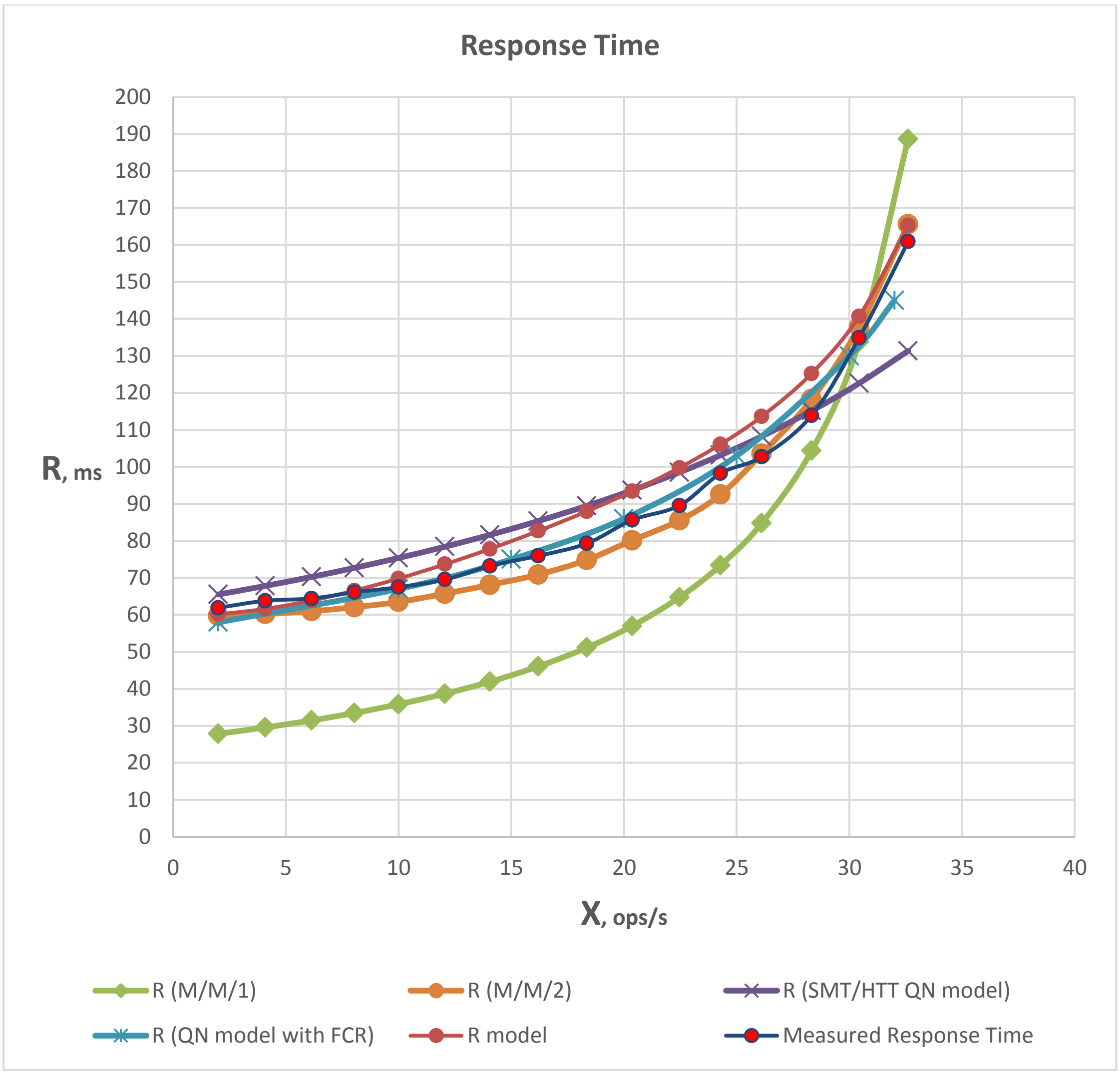

Figure 6.14: Response Time measure for 2 cores, 2660 MHz with HT

On the figure 6.14:

- For “Markov chain model”: α has been estimated to be 0,210.
- For “M/M/1”: service time s has been estimated to be 0,026.
- For “SMT/HTT QN model”: $s1$ and $s2$ are equal to 0,066ms and 0,001ms respectively.
- For “QN model with FCR”: $s1$ and $s2$ are equal to 0,055ms and 0,0001ms respectively.

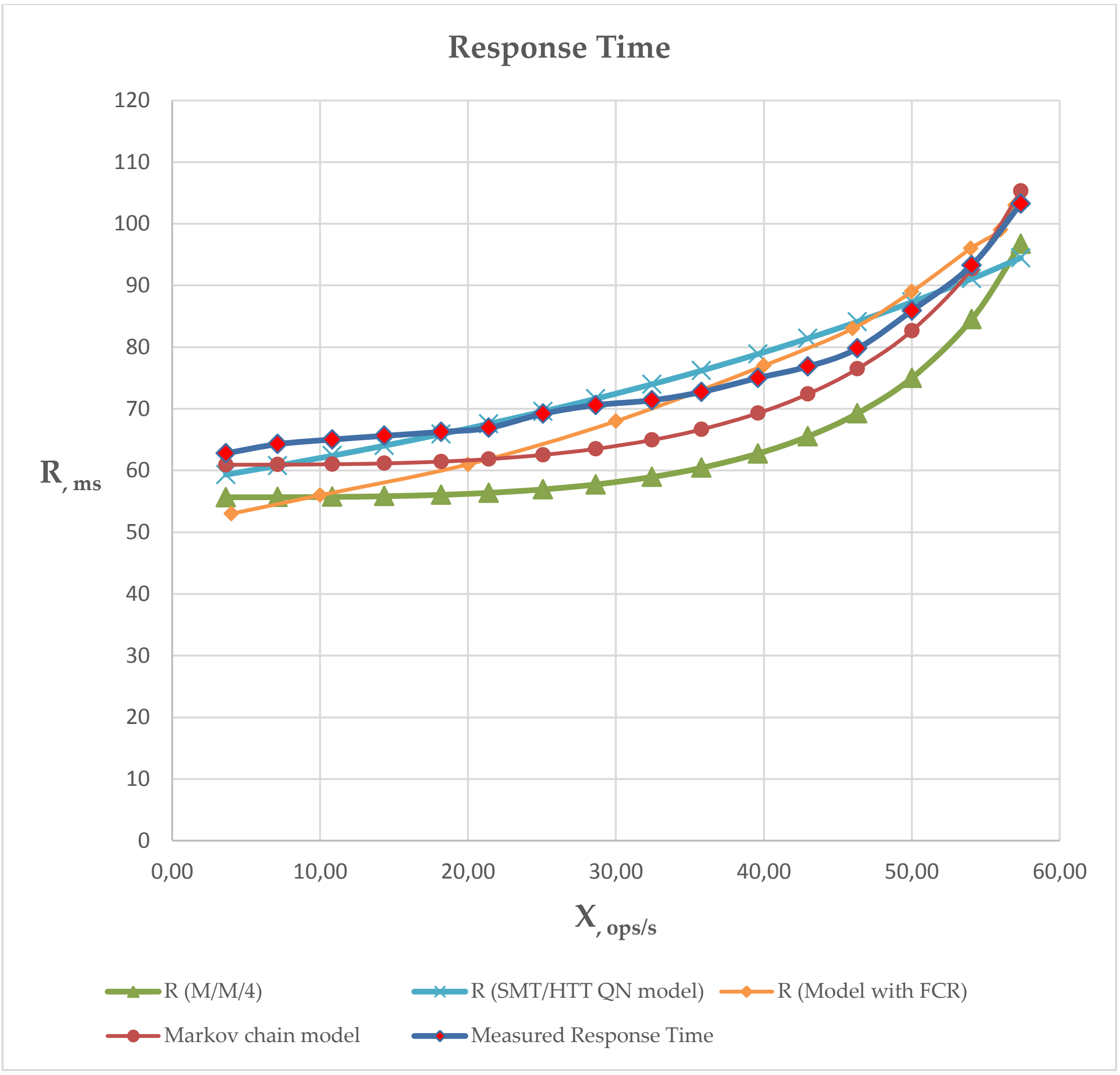

Figure 6.15: Response Time measure for 4 cores, 2660 MHz with HT

On the figure 6.15:

- For "Markov chain model": α has been estimated to be 0,095.
- For "M/M/1": service time s has been estimated to be 0,015.
- For "SMT/HTT QN model": $s1$ and $s2$ are equal to 0,055ms and 0,003ms respectively.
- For "QN model with FCR": $s1$ and $s2$ are equal to 0,052ms and 0,0001ms respectively.

Chapter 7

Conclusion and future works

In this thesis an extensive research on Simultaneous–Multithreading and Hyper–Threading technologies have been performed. Technologies that improve overall efficiency of CPU by exploitation of multiple threads that better utilize resources provided by modern superscalar processors. In most cases SMT/HTT hides memory latencies increasing throughput of computations per amount hardware used. The biggest advantage of simultaneous multithreading technique is that it requires only some extra hardware instead of replicating the entire core.

As we have experimentally validated using two different benchmarks, both technologies allow substantial performance increase sometimes reaching up to 33% over total system capacity and significant reduction of user waiting time (up to 20%). Such performance results should not be overlooked, and in fact, price and performance benefits make it a common design choice on the market. This raises another issue: Utilization law [17], which establishes linear relationship between utilization and throughput of a computing systems and which is often intended as the basis for capacity planning in large data center, as we proved experimentally, does not hold anymore in the presence of Simultaneous-Multithreading or Hype-Threading technologies. Apparently in such a case the overall capacity is heavily underestimated and there is a strong need of techniques that take into account performance effects enabled by the technologies.

In our work we propose two such models, based of birth-death Markov chain model and Queuing Network model with Finite Capacity Region. Both of them demonstrate high predictive capabilities with estimation error being within 3% to 10% and easily model SMT/HTT technologies. The proposed models are compared to other state of art models.

In the future work, another technology aspect that can be validated is power/energy efficiency of SMT. Intel claims that HTT enabled processors are extremely energy efficient across a broad range of applications. This effect is achieved due to fewer idle execution units that consume power without contributing to performance [1]. Investigations can be performed again using SPEC Power benchmark – an industry standard for measuring both the performance and the power consumption of servers [19].

Bibliography

[1] G. Drysdal, A. Valles, M. Gillespie. Intel's article: Performance Insights to Intel® Hyper-Threading Technology, 2009

[2] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty,. J. A. Miller, M. Upton. Hyper-Threading Technology Architecture and Microarchitecture, Intel Corp., 2012

[3] Intel White Paper: Next Generation Microarchitecture (Nehalem), 2008

[4] M. Bertoli, G. Casale, and G. Serazzi. JMT: performance engineering tools for system modeling. SIGMETRICS Perform. Eval. Rev., 36(4):10–15, 2009.

[5] Michael E. Thomadakis. The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms, 2011

[6] Y. Ding, Ethan Bolker1and A. Kumar. Performance Implications of Hyper-Threading. BMC Software, Inc., 2004.

[7] D. Marr, F. Binns, D.Hill, G. Hinton, D. Koufaty, J. Miller, M. Upton, "Hyper-Threading Technology Architecture and Micro architecture," Intel Technology Journal (Hyper-Threading Technology), Volume 06 Issue 01, February 14, 2002.

[8] M. Tullsen. Simultaneous Multithreading, 1996

[9] H. Wang, P. H. Wang, R.D. Weldon, S. M. Ettinger, H. Saito, M. Girkar, S. S. Liao, J. P. Shen. Speculative Precomputation: Exploring the use of Multithreading for Latency, Intel Corp., 2012

[10] W.M. Zuberek. Modeling and Analysis of Dual Block Multithreading, 2004

[11] S. D. Casey. Intel's article: How to Determine the Effectiveness of Hyper-Threading Technology with an Application, 2011

[12] G. Drysdale, M. Gillespie. Intel's White Paper: Intel Hyper-Threading Technology: Analysis of the HT Effects on a Server Transactional Workload, 2009

[13] L. W. Dowdy, D. A. Menasc´e, G. D. Virgilio, A.F. Almeida. Performance by

Design - Computer Capacity Planning by Example. Pearson Education, Inc., 2004

[14] S. P. Smith. Intel’s White Paper: Hyper-Threading: Be Sure You Know How to Correctly Measure Your Server’s End-User Response Time, 2010

[15] P.J. Denning and J.P. Buzen. The operational analysis of queueing network models. ACM Computing Surveys, 10:225-262, 1978

[16] P. E. Glenbe and J. C. C. Nelson. Introduction to Queueing Networks. John Wiley and Sons, 1987.

[17] J. Zahorjan, S.Graham, K. C. Sevcik, E.D. Lazowska. Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Inc., 1984.

[18] SPECpower_ssj2008 – User Guide, SPEC Corp.

[19] SPECpower_ssj2008 – SSJ Workload, SPEC Corp.

[20] SPECpower_ssj2008 – SSJ Control and Collect System, SPEC Corp.

[21] J.L. Hennessy, D.A. Patterson. Computer architecture – a qualitative approach” (3 ed.), Morgan Kaufman 2003.

[22] D. C. Brock. Understanding Moore’s Law: Four Decades of Innovation, 2006

[23] G. Serazzi, M. Bertoli, Giuliano Casale. User-friendly approach to capacity planning studies with java modeling tools.

[24] J.L. Hennessy and D.A. Patterson. Computer architecture, Fifth Edition: A Quantitative Approach (The Morgan Kaufmann Series in Computer Architecture and Design), 2011

[25] Java Modelling Tools - System manual,v.0.1, 9 pp, June, 2006. 2006.

[26] Java Modelling Tools- Users manual, v.0.8.0, 173 pp, Jun, 2010. 2010.

[27] jmt.sourceforge.net/. JMT website.

[28] A. Sterrett. Mathematical techniques for the performance oriented design (pod) tool. August 1990.