CORE TEMPERATURE AWARE THREAD SCHEDULING IN HETEROGENEOUS MULTI-CORE ARCHITECTURE

by

Pratik Hambirrao Jadhav

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Electrical Engineering

Charlotte

2014

Approved by:

____

Dr. Bharat Joshi

____

Dr. Ron Sass

____

Dr. Arun Ravindran

© 2014
Pratik Hambirrao Jadhav
ALL RIGHTS RESERVED

ABSTRACT

PRATIK HAMBIRRAO JADHAV. Core temperature aware thread scheduling in heterogeneous multi-core architecture.
(Under the direction of DR. BHARAT JOSHI)

Along with the evolution of heterogeneous multi-core architectures, advanced thread scheduling algorithms for maximizing system efficiency are needed. As power density increases with technology, core temperature variations can cause bottlenecks in the system performance. More efficient thread scheduling algorithms for heterogeneous multi-core architectures and advanced dynamic thermal management (DTM) techniques are needed in the current multi-core era.

The performance and power requirements of a thread vary significantly during its execution. Threads can be reassigned to cores upon detection of such variations. This can improve the power efficiency and performance of the heterogeneous multi-core architectures. Traditional online learning schemes rely on sampling to determine the thread-to-core affinity. However, as the number of cores increase, the sampling overhead would be high [3]. Moreover, the core temperature must be below a safe level, or else significant power dissipation can occur. Overheating of the cores can cause hot spots which severely reduce their lifespan and may affect the runtime performance.

The objective of this thesis is to analyze a dynamic thermal management technique, which considers core temperature variations for heterogeneous multi-core architectures. While doing so, the thread scheduling decisions in heterogeneous multi-core architectures are taken using an online estimation technique. A scheduler is proposed which can estimate the power and performance of the current thread on all the cores. Depending upon the expected power and performance requirements, the appropriate thread-to-core pair is selected by the scheduler. A thermal profiler is proposed which classifies preempted threads into hot, warm and cold categories. These threads are classified according to the temperature variations which they produce on any particular core. An additional task of rescheduling these classified threads is performed by the scheduler, considering the core temperatures. The queuing network model is used to analyze the proposed design.

DEDICATION

I would like to dedicate this Thesis to my parents Mrs. Sudha Jadhav and Mr. Hambirrao Jadhav and to my dearest sister Miss. Prajkta Jadhav. Their love, support and faith kept me motivated and their constant encouragement helped me in completing my task. My parents have always given me freedom to follow my heart and just the thought of them is enough to guide me in difficult phases.

ACKNOWLEDGMENTS

"Happiness does not come from doing easy work, but from the afterglow of satisfaction that comes after the achievement of a difficult task that demanded our best."
- Theodore Isaac Rubin

This thesis wouldn't be possible without my advisor Dr. Bharat Joshi, who introduced me the world of Computer Architecture and Performance Evaluation of computer systems. His constant support at every stage and invaluable advice brought best out of me and above all, I appreciate his patience when I was making mistakes.

I have also learned a lot from my mentors Dr. Ron Sass and Dr. Arun Ravindran, they were always forthcoming with advice on research and career. I would also like to thank all my mentors, professors, staff of the Electrical and Computer Engineering Department.

Chinmay, Akshay, Rohan, Aditya, Apurva, Abhishek, Sameer and Mrunal, you guys never made me feel that I had left home when I first arrived in Charlotte. Vinmay, Rishi, Anand, Yashada, Vinit, Chinmay Jr., Akriti : I owe you guys much more than you realize. Special note of thanks to my special friends Mayur Tare, Ninad Vartak, Saptarshi Banerjee, Sumedh Utpat, Kaveri Inamdar, Ameya Lohokare, Aniket Bhosale and Mrunal Damle. You people have been a constant source of strength.

TABLES OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

In 1965, Gordon E. Moore, co-founder of the Intel Corporation proposed a law that the number of transistors on a chip increases exponentially, doubling approximately every 2 years. The processor performance also increases accordingly. Increase in power density causes high temperatures, which threaten the system performance and safety [2]. High temperatures in the system, increases the power leakage, reduces the speed of the device and finally it increases the cooling costs. Moreover, hot spots and steep temperature gradients of the cores have a severe effect on the reliability and lifespan of the system [2]. It is very important to design thermal management techniques which can balance the temperature distribution and reduce hot spots.

Power density concerns in processor ICs led to the multi-core era [6], where a single and very high power processor was replaced by several smaller cores having more modest computational capabilities. In homogeneous multi-core processors, incoming program threads/tasks can be assigned to cores arbitrarily. Advanced research in this area, gave rise to the heterogeneous multi-core processors (HMPs). For a given power budget, homogeneous multi-core processors have been outperformed by heterogeneous multi-core processors. This is achieved by well balanced thread-to-core assignments in the HMPs. Also, HMPs provide better area-efficient coverage of workload demands.

There have been a number of proposed thread scheduling schemes for HMPs such as, offline profiling, online learning via program sampling [12, 13, 14, 15]. In the proposed model, a method similar to the method proposed in [3] is considered. Both the performance and power of a thread is estimated online on all the cores in the HMP using performance counters. Using these estimated values, the thread-to-core pair is selected. Performance counters like L2 miss, TLB misses, number of fetched instructions, IPC, power, retired INT, L1 hit and dispatch stalls are used to determine thread-to-core affinity. These counters show high correlation to power and performance of the core [3]. This information is provided to the scheduler which will take the thread scheduling decisions. Using this thread scheduling technique, better thread-to-core assignments can be achieved which overcome the drawbacks of offline profiling and online learning.

Temperature of the cores is an important factor which needs to be taken into consideration while studying the techniques for performance and power improvement in the system. Issues related to the core temperature variations can be eliminated by proposing a new technique of dynamic thermal-aware thread scheduling which collaborates with the OS and architecture [2]. The temperature is directly proportional to the leakage current, and so techniques that minimize the overall temperature are highly beneficial in terms of saving energy and power [1].

In the past, several techniques have been proposed for dynamic thermal management (DTM) such as, fetch throttling, dynamic voltage and frequency scaling (DVFS) [8, 9, 10]. However, these methods are reactive in nature i.e., they take action only after temperature of a core reaches a certain level. These methods are effective for cooling on-chip hot spots, but they result in performance degradation. They do not target reducing the temperature as much as possible to minimize the impact on reliability [2]. However, proactive DTM techniques are more effective in the temperature control against thermal emergency, since they trigger the control schemes before the core temperature reaches the emergency threshold. In one of the proactive DTM technique, the thread temperature characteristics are profiled and threads are divided into three categories, that are cold threads, warm threads and hot threads [2]. There is also a cool loop to reduce the temperature of the processor core. Threads are classified depending upon the variation of the core temperature. The scheduler will take decisions about rescheduling the threads which are increasing the temperature of the core beyond a predefined threshold, while the thermal profiler will classify the threads into the three categories. Information about the core temperature variations will be provided to the scheduler as well as the thermal profiler by the thermal sensors which are placed on each of the heterogeneous cores. Together, the scheduler and the thermal profiler will take care of the thread scheduling, thread migration, thread classification and thread rescheduling. The scheduler acts at the OS level, taking decisions about the thread-to-core assignments, thread migrations and thread rescheduling. The thermal profiler will take decisions about the thread classifications.

An algorithm is designed considering the above mentioned techniques of online estimation for thread scheduling in HMPs and proactive DTM. An analysis of this algorithm is done using the method of queuing theory. The queuing system is the simplest way of performance modeling of any computer system. In the proposed model, the incoming threads are defined in terms of the arrival rates and the cores are treated as the servers having different service rates. Threads will either leave the system upon service completion or will get preempted and rescheduled through the thermal profiler due to variations in the core temperature. This representation is called as a queuing network.

The open network queuing model for the proposed algorithm is shown in Figure 1.1. In the proposed model, three heterogeneous cores are considered for the execution of the threads. All three cores will have different service rates. All the incoming threads (internal as well as external) will arrive in the priority queue. The scheduler is placed between a priority queue and heterogeneous cores. The scheduler will take thread scheduling decisions for all the arriving threads. The preempted threads and the threads which have increased core temperature beyond predefined threshold, will arrive at the thermal profiler. These threads will get profiled and classified into the hot/warm/cold queues. The threads from hot/warm/cold queues are treated as the internal threads and the threads from ready queue are treated as the external threads. The thermal profiler will behave as a single-server-single-queue model while the scheduling unit will behave as a heterogeneous-multi-server-single-queue model with preemptive thread scheduling. The detailed steady state analyses and the calculations of response times and queuing lengths are done in chapters 3 and 4.

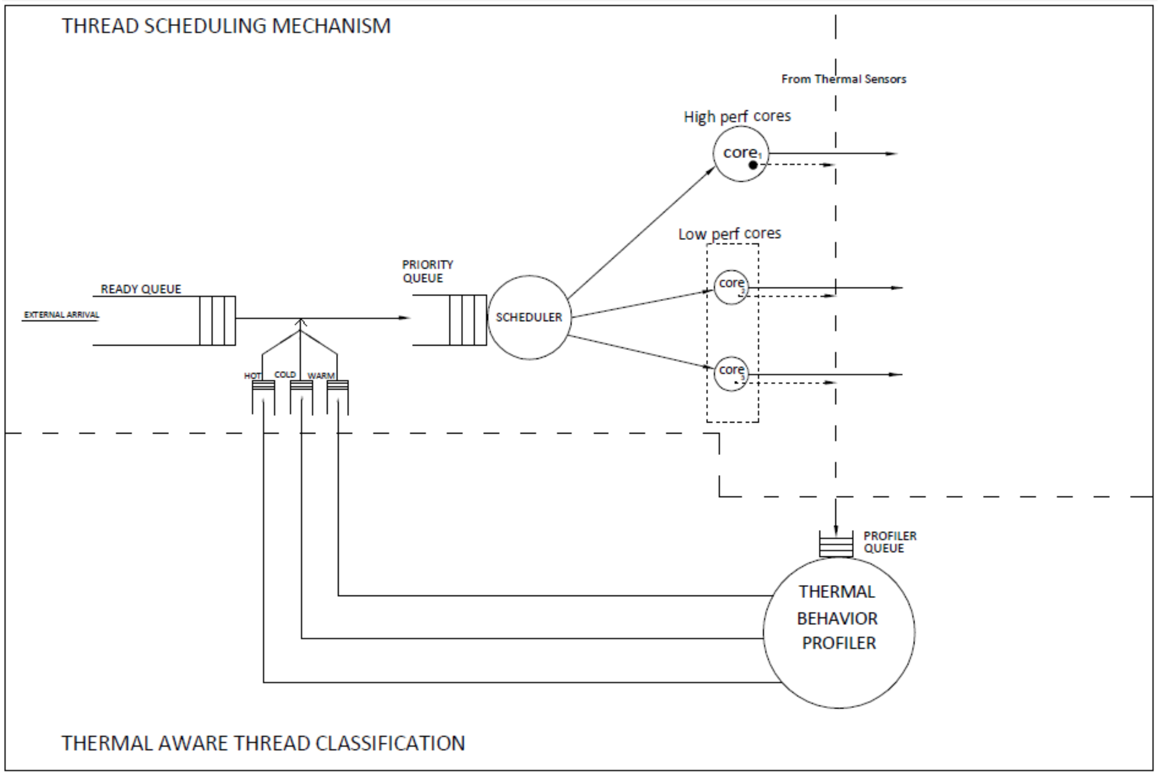


Figure 1.1: Queuing model of the proposed system

CHAPTER 2: RESEARCH BACKGROUND

In this chapter, an overview of the previous work falling within the scope of this thesis is presented. The aim of this chapter is to give the reader an idea about the current state of the art in the field of thread scheduling on HMPs. Also, this chapter will give a survey of thermal-aware thread scheduling techniques which can be implemented for heterogeneous as well as homogeneous multi-core architectures. An overview of performance evaluation of a computer system using queuing theory is also presented in this chapter.

2.1 Thread Scheduling in Heterogeneous Multi-core Processors (HMPs)

Several thread scheduling algorithms have been proposed for HMPs. These algorithms are broadly classified as offline profiling, online learning via sampling and online estimation. Limiting the systems to rely on manual or static mapping of threads to cores in an asymmetric multi-core causes the performance loss in HMPs. Hence, there was a need of online learning or online estimation techniques for thread-to-core assignments.

There have been number of techniques based on offline profiling in HMPs. In [12], a technique was proposed which uses regression analysis along with phase classification to identify the thread-to-core affinity. HASS: Heterogeneity-Aware Signature Supported scheduling algorithm presented in [13], used architectural signatures, i.e., compact summary of the thread's architectural summary collected offline. There is one more paper which used multi-dimensional curve fitting to determine thread-to-core affinity offline [24]. All these approaches relied on offline profiling i.e., these methods require prior knowledge of the workloads that will run on the multi-cores, which is not practical as we can't predict the thread's resource demands.

Online learning based schemes are a practical alternative to the offline profiling schemes. Whenever a new thread is running and a new phase [14] is detected, sampling is initiated and the core which provides the better power efficiency is chosen, assuming that the system consists of cores of various sizes and all are running the same ISA [15]. A similar approach for thread scheduling was proposed by [16] where the thread swapping between the two types of cores is performed depending upon the detection of a phase change. Several algorithms were proposed in [29] for power management in HMPs via thread scheduling using program sampling. But all these online learning schemes have a limitation. Whenever there is an increase in the number of cores in the system, for each phase detected, the number of samples will be large and these schemes will experience a high overhead.

To avoid sampling and the resulting overhead, online estimation techniques are better solutions. Here, depending upon the current characteristics of a thread which is being executed, its performance on other core types is estimated. Accuracy of the estimation determines the benefits of the scheme. In [17], closed form expression was used to estimate the performance of the thread currently running on all the cores. The disadvantage of this approach was that they considered specific cores and lacked a general approach. The same disadvantage was also observed in [18] where thread scheduling was done using predetermined rules and also very specific cores were considered.

Overall, many scheduling schemes are available, out of which the estimation-based schemes are the most practical and scalable, as these schemes don't require prior knowledge about the computational needs of the different workloads. Since, sampling is not needed, there is no sampling overhead as well. In the proposed model, a method similar to the method proposed in [3] is used, where the performance counters like L2 miss, TLB misses, number of fetched instructions, IPC, power, retired INT, L1 hit and dispatch stalls are used to determine the thread-to-core affinity. Also, these counters focus not only on the performance improvement, but also the improvement in power utilization of the cores. Having the same set of performance counters for all the core types (heterogeneous cores) and for both the metrics (performance and power) greatly simplifies the estimation mechanism [3].

In the proposed algorithm, thread scheduling/allocation is handled by the scheduler. It is considered that the scheduler has knowledge about the availability of the cores and also has the capability of online estimation of the thread's power and performance on the various cores. This online estimation technique helps the scheduler in thread allocation by selecting a suitable thread-to-core pair. Additionally, the scheduler must be aware of the core temperature variations (hot/warm/cold cores) as the scheduler has to perform an additional task of core temperature aware scheduling. Threads are received from the ready queue (external arrival), hot queue, warm queue and cold queue.

2.2 Dynamic Thermal Management Techniques for Thread Scheduling

Over the years, many techniques were proposed to control and manage the temperature of the core, such as dynamic temperature management (DTM) techniques. DTM is a common technique used to keep the core temperature in the safe region. DTM techniques deal with the threshold temperature and the trigger temperature. When the temperature of a chip arrives at the trigger temperature, the DTM technique is triggered. Moreover, if the temperature of a chip is over the threshold temperature, it shows that the system is in a critical condition.

In [19] a coordinated hardware-software approach based for DTM was proposed in which, software techniques dynamically adjust process priorities based on their estimated individual temperatures. Also, depending upon the priorities, they assign shorter time slices to the 'hot' processes and longer time slices to the 'cold' processes. But this method had a major drawback, as the 'hot' processes will receive a shorter time slice, they will take longer to complete. In [20], a framework was proposed in which by proactive task migration among neighboring cores, distributed thermal management can be achieved.

Thermal-aware scheduling techniques (software based) also play a vital role in the thermal management problem. For example in [21], a trade-off between the temporal and spatial hot spot mitigation schemes and thermal time constants can be reduced by lowering the on-chip unit temperature with the change in the workload in a timely manner with the OS and hardware support. OS-level dynamic scheduling policies were designed and evaluated in [22]. But this method used a random policy for sending workload to the cores.

A thermal management technique which combines both the proactive and reactive schemes to prevent high temperatures was proposed in [2] where, threads were classified based on their temperature characteristics and assigned to the cores according to the core temperatures. On similar lines, in the present work, the task of thread migration, thread classification and thread reallocation is collectively done by the scheduler and the thermal profiler. Thermal sensors placed on each core will provide information about the core temperature variations to the scheduler as well as the thermal profiler. Using this information, the scheduler will schedule the threads on the cores while the thermal profiler will classify the threads into hot/warm/cold queues. In the previous studies, threads were often classified into two categories of hot and cold. To obtain more accuracy, a more detailed classification of the threads must be used. Therefore, the threads are classified into three categories such as, hot threads, warm threads and cold threads. A detailed implementation of this logic is explained in chapter 3.

2.3 Queuing Models

The analytical queuing theory model is the simplest technique for the computer system performance modeling. The queuing network models achieve relatively high accuracy at relatively low cost [23]. Queuing theory talks about the analysis of waiting lines and it is a basic tool in the analysis of the complex computer systems. The queuing model is an abstract description of any system. In the proposed model, steady state analyses of standard queuing models are used to determine mean response time and average queue length of the entire model. Using these two performance measures the efficiency of the proposed algorithm is determined.

CHAPTER 3: DEVELOPMENT OF ALGORITHM AND ANALYTICAL DERIVATIONS FOR PROPOSED MODEL

The first section of this chapter gives the detailed functionality of the thermal profiler. The thermal profiler classifies the threads into hot, warm and cold queues according to the thread's temperature characteristics and core temperature variations. The second section of this chapter explains the functionality of the scheduler which schedules/reschedules the threads on the heterogeneous multi-cores. The third section talks about the algorithm for the proposed thesis. In the last section, the steady state analysis and derivations are presented for the standard queuing models. Using these equations the average response times and average queue lengths are calculated for the proposed queuing model.

3.1 Thermal Profiler

The thermal profiler uses a proactive dynamic thermal management technique. It takes the core's runtime temperature and the thread's temperature characteristics into consideration to take scheduling decisions. This scheme is very useful for controlling the core temperature variations and hot spots [2]. According to the temperature characteristics of the threads, they are classified into three categories, cold threads, warm threads and hot threads. Whenever a thread running on any of the cores, increases the temperature of that core above the high temperature threshold, it is classified as a hot thread and whenever preempted by the scheduler it must be placed in the hot queue. Whenever the thread which is executed on any of the cores, maintains the temperature of that core below the low temperature threshold, it is classified as a cold thread and whenever preempted by the scheduler it must be placed in the cold queue. All other threads which maintain the temperature of the core between the high threshold and low threshold are classified as warm threads and once preempted by the scheduler, these threads must be placed in the warm queue. Warm thread queue is used as a buffer zone to prevent the drastic changes in the temperature of the cores.

The scheduler preempts the low priority threads executing on the heterogeneous cores. It also preempts the threads which have increased the current temperature of the core beyond a predefined threshold. All these preempted threads will come to the thermal profiler via a profiler queue. The thermal profiler will receive information from the thermal sensors about the rising temperature of the core caused by any of the threads. The thermal profiler's task is to classify the threads. The thermal profiler will follow algorithm given below:

Assume that,

t_i - rising temperature caused by thread i

θ_h - high temperature threshold of t_i

θ_l - low temperature threshold of t_i

```
**switch** $t_i$ **do**
    **case** $t_i > \theta_h$
        enqueue the thread to hot queue
    **case** $t_i < \theta_l$
        enqueue the thread to cold queue
    **otherwise**
        enqueue the thread to warm queue;
    **end**
**endsw**
```

The thermal profiler will take decisions about the classification of threads as per the above algorithm. These classified threads are placed in the hot/warm/cold queue. The thermal profiler tags each thread as hot/warm/cold queue thread. These tags will help the scheduler in assigning threads to the suitable cores based on the temperature of the cores. For example, if a thread is from the hot queue, it will be executed on a core with low temperature.

3.2 Scheduler

The basic task of the scheduler is to ensure that the incoming threads get scheduled on the cores depending upon their resource requirements. All external (ready queue) and internal (hot/warm/cold queue) threads will come to the priority queue. The scheduler will receive the threads from the priority queue. The scheduler schedules the threads on the cores depending upon their performance and power requirement.

The scheduler will also maintain a table in which it will store the information about the individual core's temperature. The scheduler will get this information from the thermal sensors placed on each of the cores. It will reschedule the threads considering their temperature characteristics and individual core's temperature. The scheduler will work at the OS level.

T_{Ci} - Temperature of core i

θ_{Cl} - low temperature threshold of core

θ_{Ch} - high temperature threshold of core

Algorithm 1 Scheduler Algorithm

```
if $T_{Ci} > \theta_{Ch}$ then
    schedule cold thread
    if cold thread not available then
        execute cool loop and simultaneously start execution of warm thread or ready
        queue thread
    end if
    if warm thread or ready queue thread not available then
        execute cool loop. allow temperature of the core to drop below predefined
        threshold
        start execution of any available thread
    end if
else if $T_{Ci} < \theta_{Cl}$ then
    schedule hot thread
    if hot thread not available then
        execute warm thread or any thread from ready queue
    end if
end if
```

All the threads from the external queue, warm, hot and cold queue will come into the priority queue. The significance of using a priority queue are:

- In a priority queue, an element with a high priority is served before an element with a low priority. If two elements have the same priority, they are served according to their order in the queue i.e., first-come-first-serve (FCFS).
- Also, threads coming to the priority queue will have a tag which will tell the scheduler from which queue they have arrived (hot/warm/cold/ready queue). This will help the scheduler in scheduling threads on cores depending upon the thread's category and core's temperature characteristics.
- Although preemptive thread scheduling is used, use of a priority queue will help in reducing the number of preemptions.
- Only two types of threads: High priority and Low priority threads.

Assumptions related to the scheduler:

1. All the arriving threads will form a single waiting line and they are served as per the priority queue rules mentioned above.
2. If all the cores are idle, the first thread will get scheduled on the fastest core.
3. If parts of the cores are idle, the next thread will get scheduled on the faster core.
4. If all cores are busy, the next arrived thread waits for scheduler's decision. The scheduler will take decisions of thread scheduling.
5. Whenever a thread gets preempted or whenever it increases the temperature of the core beyond predetermined threshold, the scheduler will send that thread to the thermal profiler.

Algorithm 2 Functionality of the scheduler

```
if All the cores are busy then
    - Check priority of the arriving thread
    if Arriving thread is of high priority then
        if A core executuing low priority thread is found then
            - The scheduler will preempt that thread
            - The preempted thread will go to the thermal profiler
            - The arrived thread will gets scheduled on that core
        end if
    end if
    if Two cores are executing low priority threads then
        - Take performance requirement of the arriving thread into consideration
    end if
    if Arriving thread is high or low priority cold thread then
        - The scheduler will schedule it on the hot core
        if No cold thread in waiting then
            - Check for warm queue thread
        end if
        if No warm thread in waiting then
            - Start the execution of cool loop to reduce temperature of the hot core
            - Once the temperature drops to a present colder temperature, a suitable
              thread will be assigned
        end if
    end if
    if Arriving thread is from hot queue and of high priority then
        if Low priority threads are executing on the cores then
            - Thread's performance requirement will be taken into consideration
        end if
        if All cores are hot then
            - A cool loop will be executed on the core
            - Once temperature of the core falls within acceptable range, hot thread will
              start its execution
        end if
    end if
    if Arriving thread is from cold queue and of high priority then
        if The scheduler finds two hot cores with low priority threads then
            - Check for performance requirement of the thread
            - Schedule that particular thread on suitable hot core
        end if
    end if
end if
```

3.3 Thermal-aware and Preemption based Thread Scheduling Algorithm

Important characteristics of the proposed algorithm:

1. When a thread arrives from the ready queue, the scheduler will decide the performance requirement of that particular thread.
2. Depending upon the availability of the cores as well as the priorities associated with the threads, the scheduler will schedule them on the high performance (HPerf) cores or low performance (LPerf) cores. Threads which require HPerf cores can be scheduled on the LPerf cores and vice a versa.
3. If a high priority thread comes to the scheduler it will preempt already executing thread (it might be on HPerf or LPerf cores).
4. A cool loop is used to prevent overheating. While the cool loop is executed, core temperature will be checked for redistributing threads as quickly as possible.
5. Information of all the cores and their temperatures will be available to the scheduler.

Algorithm 3 Thermal-aware and preemption based thread scheduling algorithm

```
if Ready queue is not empty then
    - Thread will be placed in a priority queue
    - Check for the resource requirement, find the suitable thread-to-core pair
    - Depending upon the resource requirement and availability of the core, this
    thread will be scheduled
    if All the cores are busy and high priority thread arrives then
        - Preempt the core which is executing low priority thread
        - Depending upon the core temperature variations preempted thread will be
        placed in hot, warm or cold queue via thermal profiler, from hot/warm/cold
        queue it will be directly come to priority queue
        - At scheduler this thread will get rescheduled depending upon its new require-
        ment and priority associated with it.
    end if
    - Monitor temperature variations of all the cores by keeping a track of thermal
    sensors
    - The scheduler checks for the performance requirement of each thread
    if Core temperature of any core goes above high threshold then
        - Move that thread into hot queue via thermal profiler
        - Check for any thread from cold queue present in priority queue
        if Cold thread not available then
            - Start executing cool loop and start execution of warm thread or ready
            queue thread
        end if
        if Warm thread not available then
            - Start executing cool loop and allow core temperature to drop below prede-
            fined threshold
            - As per the scheduler, start the execution of any available thread on this
            core
        end if
        if Thread moved to hot queue is high priority thread then
            - It will arrive in priority queue via thermal profiler and it can preempt any
            core executing low priority thread and having low core temperature variations
        end if
    end if
end if
```

Some special cases :

- If one of the LPerf cores is hot and other one is busy executing some high priority thread, that point of time if high priority thread arrives requiring low performance cores it can preempt high performance core which is executing low priority thread.

- If we have a hot core and we don't have any cold/warm thread, we need to run a cool loop on that particular core. While running a cool loop; we can execute the low priority hot thread/any priority warm thread/any priority cold thread. There will be performance degradation, but none of the cores are kept idle.

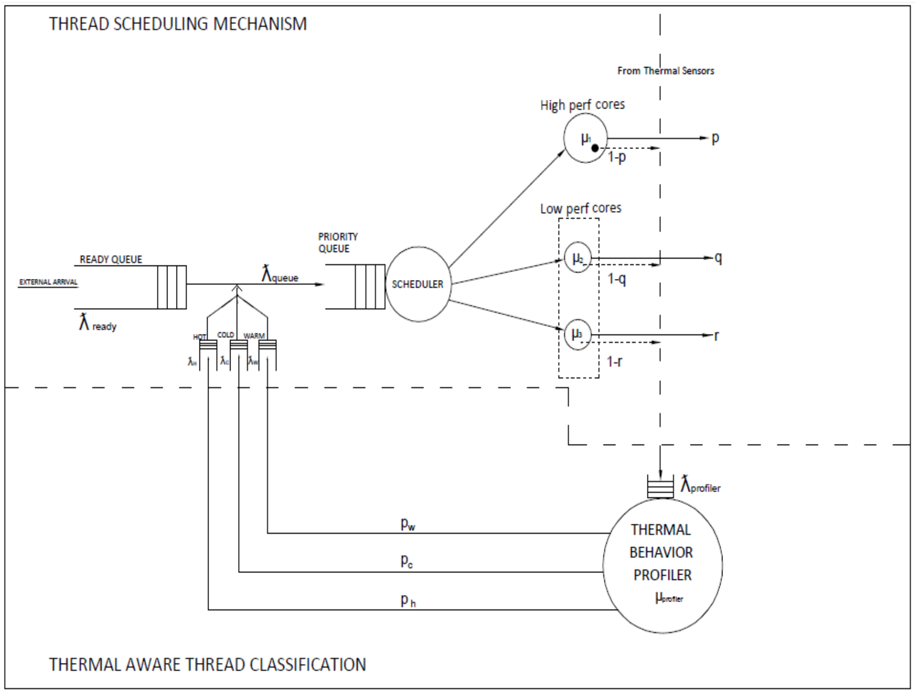


Figure 3.1: Queuing model with probabilities, arrival rates and service rates assigned

Figure 3.1 shows proposed queuing model with probabilities, arrival rates and service rates assigned. We will be using these assigned values in the derivations of average queue lengths and mean response time of the system.

3.4 Derivations for Standard Queuing models

From the above Figure 3.1, the proposed queuing model is divided into two parts: thread scheduling mechanism and thermal-aware thread classification. In the thread scheduling mechanism, all the threads (external as well as internal) will arrive at a single priority queue. The scheduler will schedule these threads on three heterogeneous cores. Upon completion of the task, these threads will exit the system. All the preempted threads will come to thermal profiler. In queuing theory, this model can be represented as M/M/3 heterogeneous cores queuing model. In thermal-aware thread classification section, the preempted threads will come to the thermal profiler. They will form a single-queue at the profiler. Here, threads will get classified into three categories. We can represent this part of the system as M/M/1 queuing model. We will perform steady state analysis for these two models and we will derive the average queue length and mean response time for them.

M/M/1 and M/M/3 notations can be explained using Kendalls notation.

A queuing model can be described using a notation which uses a series of symbols and slashes such as A/B/X/Y/Z, where

A: indicates the inter-arrival time distribution

B: indicates the service time distribution

X: the number of parallel cores

Y: the restriction on system capacity

Z: the queue discipline

More often a shorter notation, A/B/c, is used when there is no limit on the waiting line, and the queue discipline is FCFS. The standard symbols used for A and B are G, general service times; Ek, Erlang-k inter-arrival or service time distribution; M, exponential inter-arrival or service time distribution; D, deterministic (constant) inter-arrival or service time distribution.

In queuing theory, the most common models assume that the inter arrival times and service rates have exponential distributions i.e., their arrival rates and service rates follow a Poisson distribution. If the probability that the process will go from the i-th state to j-th state depends only on the probability of the present state i, then the process is called Markov chain. Poisson process satisfies the Markovian property.

In the proposed queuing model, consider that arrivals and departures are independent and exponential in nature i.e., arrival rates and service rates are Markovian in nature. Hence, in Kendall's notation form we will represent our models as M/M/1 queuing model and M/M/3 heterogeneous cores queuing model.

For both the models, M/M/3 heterogeneous cores queuing model and M/M/1 queuing model we have used single-line queue. Advantages of using single-line queue:

- First-come-First-served : Everyone has an equal wait. Fairness (no favoritism or 'without getting lucky'). No rushing.
- Individual wait times in a single-line queue are significantly lower. No one gets stuck behind slow poke!
- Avoids line switching (jockeying).

3.4.1 M/M/1 Queuing Model

M/M/1 is the simplest queuing model with just a single core. In the steady state, we consider that the inter-arrival times and service times are exponentially distributed. Models with Markovian arrivals and service rates can be explained using the birth-death processes.

λ - arrival rate

μ - service rate

n - number of threads in the system

ρ - traffic intensity $= \frac{\lambda}{\mu}$

p_n - probability of having n threads in the system

p_0 - probability of having 0 threads in the system

The flow-balance equations for this system are given by [26]:

$(\lambda + \mu)^* p_n = \mu^* p_{n+1} + \lambda^* p_{n-1}$ (n > 1)

$\lambda^* p_0 = \mu^* p_1$

Iteratively, above equations give

$p_n = (\frac{\lambda}{\mu})^n p_0 = \rho^n p_0$

$p_0 = 1$ - $\frac{\lambda}{\mu}$

Using the above two probabilities values, it's easy to calculate the average number of threads in the M/M/1 system, waiting time (response time) in the system and the average queue length. The direct equation for these parameters are presented in [27]. L_s - average number of threads in the system L_q - average number of threads in the queue W_s - average waiting time (response time) in the system

$$L_s = \frac{\rho}{(1-\rho)} \tag{3.1}$$

$$L_q = \frac{\rho^2}{(1-\rho)} \tag{3.2}$$

$$W_s = \frac{1}{(\mu-\lambda)} \tag{3.3}$$

3.4.2 M/M/3 Queuing Model with Heterogeneous Cores

Steady state analysis of M/M/3 with ordered heterogeneous cores system was previously done in [28] and [29]. The average waiting time in a system and the average queue length for this system is caluclated as follows:

λ - Poisson arrival rate

μ_1, μ_2, μ_3 - mean service rates

Assume that,

- Arriving threads form a single queue and they are served on FCFS basis.
- If all cores are idle, first thread goes to the fastest core.
- All cores are busy, the thread will wait in the queue until any one core becomes free.

We will consider, $\mu_1 > \mu_2 > \mu_3$.

p_n - probability of having n threads in the system

p_0 - probability of having 0 threads in the system

The steady-state equations for this system are given by [29] :

$\lambda * p_0 = \mu_1 * p_1$

$(\lambda + \mu_1)^* p_1 = (\mu_1 + \mu_2)^* p_2 + \lambda^* p_0$

$(\lambda + \mu_1 + \mu_2)^* p_2 = (\mu_1 + \mu_2 + \mu_3)^* p_3 + \lambda^* p_1$

$(\lambda + \mu_1 + \mu_2 + \mu_3)^* p_n = (\mu_1 + \mu_2 + \mu_3)^* p_{n+1} + \lambda^* p_{n-1}$, $n \geq 3$

Solving recursively,

$$p_n = \begin{cases} (\frac{\lambda}{\mu_1}) * p_0, & n = 1 \\ (\frac{\lambda^2}{\mu_1 * (\mu_1 + \mu_2)}) * p_0, & n = 2 \\ (\frac{\lambda^3}{\mu_1 * (\mu_1 + \mu_2) * (\mu_1 + \mu_2 + \mu_3)}) * p_0, & n = 3 \end{cases} \tag{3.4}$$

Equation 3.4 can be modified for two different cases of n as,

$$p_n = \begin{cases} (\frac{\lambda^n}{\prod\limits_{k=1}^{n} \sum\limits_{j=1}^{k} \mu_j}) * p_0, & 1 \leq n \leq 2 \\ (\frac{\lambda^3}{\prod\limits_{k=1}^{3} \sum\limits_{j=1}^{k} \mu_j}) * (\rho^{n-3}) * p_0, & n \geq 3 \end{cases} \tag{3.5}$$

And,

$$p_0 = \left[1 + \sum_{n=1}^{2} \frac{\lambda^n}{\prod\limits_{k=1}^{n} \sum\limits_{j=1}^{k} \mu_j} + \frac{\lambda^3}{\prod\limits_{k=1}^{3} \sum\limits_{j=1}^{k} \mu_j * (1 - \rho)} \right]^{-1} \tag{3.6}$$

Let, E[N] denote the expected number of threads in the system.

E[N] $= \sum\limits_{n=1}^{\infty} n.p_n$

Using equation 3.5,

$$E[N] = \left[\sum_{n=1}^{2} \frac{n * \lambda^n}{\prod\limits_{k=1}^{n} \sum\limits_{j=1}^{k} \mu_j} + \frac{\lambda^3}{\prod\limits_{k=1}^{3} \sum\limits_{j=1}^{k} \mu_j} * \sum_{n=3}^{\infty} n * \rho^{n-3} \right] * p_0 \tag{3.7}$$

To simplify the last term of the above equation, assume that, $n - 3 = i$

$\sum\limits_{n=3}^{\infty} n * \rho^{n-3}$

$= \sum_{i=0}^{\infty}(i+3) * \rho^i$

$= \frac{\rho}{(1-\rho)^2} + 3 * \frac{1}{(1-\rho)}$

$= \frac{3-3*\rho+\rho}{(1-\rho)^2}$

$$E[N] = \left[\sum_{n=1}^{2} \frac{n * \lambda^n}{\prod_{k=1}^{n}\sum_{j=1}^{k}\mu_j} + \frac{\lambda^3}{\prod_{k=1}^{3}\sum_{j=1}^{k}\mu_j} * \frac{3-3*\rho+\rho}{(1-\rho)^2} \right] * p_0 \tag{3.8}$$

Using Little law, waiting time in the system, $W_s = \frac{E[N]}{\lambda}$

$$W_s = \frac{1}{\lambda} * \left[\sum_{n=1}^{2} \frac{n * \lambda^n}{\prod_{k=1}^{n}\sum_{j=1}^{k}\mu_j} + \frac{\lambda^3}{\prod_{k=1}^{3}\sum_{j=1}^{k}\mu_j} * \frac{3-3*\rho+\rho}{(1-\rho)^2} \right] * p_0 \tag{3.9}$$

Now, Average queue length of system,

$L_q = \sum_{i=3}^{\infty}(i-3) * p_i$

From equation 3.5 ,

$L_q = \sum_{i=3}^{\infty}(i-3) * \frac{\lambda^n}{\prod_{k=1}^{3}\sum_{j=1}^{k}\mu_j * (\mu_1+\mu_2+\mu_3)^{i-3}} * p_0$

$L_q = p_0 * \frac{(\mu_1+\mu_2+\mu_3)^3 * \rho^3}{\prod_{k=1}^{3}\sum_{j=1}^{k}\mu_j} * \sum_{i=3}^{\infty}(i-3) * \rho^{i-3}$

Substitute $i-3=k$,

$\sum_{k=0}^{\infty} k * \rho^k = \frac{\rho}{(1-\rho)^2}$

This gives final equation of the average queue length,

$$L_q = p_0 * \frac{(\mu_1+\mu_2+\mu_3)^3 * \rho^4}{\prod_{k=1}^{3}\sum_{j=1}^{k}\mu_j} * \frac{1}{(1-\rho)^2} \tag{3.10}$$

In the next chapter, we will use the above derived equations to calculate the average queue lengths and average response time for the proposed queuing model.

CHAPTER 4: RESULTS

Steady state analysis parameters such as, the response time and queue length are the most significant parameters for any queuing model. These two parameters are used for the performance analysis of computer systems. They define the behavior of any queuing system. Following are the six basic characteristics of a queuing model :

1. Arrival pattern of customers
2. Service pattern of customers
3. Queue discipline
4. System capacity
5. Number of service channels
6. Number of service stages

Queuing model for the proposed work is presented in Figure 3.1. It is classified in two sections :

1. Thread scheduling mechanism (M/M/3 heterogeneous cores model with preemptive priority scheduling)
2. Thermal aware thread classification (M/M/1 model)

In the following sections, the derivations from chapter 3 are applied to the proposed queuing model to get the actual response time and the average queue length for the incoming threads.

4.1 Thread Scheduling Mechanism

In this stage, threads coming from the external source (ready queue) and internal sources (hot/warm/cold queue) are combined together and sent to the priority queue. The scheduler takes care of the scheduling and rescheduling of the threads. It decides the performance requirement of a particular thread keeping track of the core temperature variations. Consider that, the scheduler operates at the OS level and it just assigns the thread depending upon core availabilities, hence the scheduler's response time is not taken into consideration. Only service rates of heterogeneous cores are taken into consideration. At this stage, time taken by the threads on heterogeneous cores and queue length of a priority queue is determined as follows:

λ_{ready} - arrival rate from ready queue

λ_H - arrival rate from hot queue

λ_C - arrival rate from cold queue

λ_W - arrival rate from warm queue

λ_{queue} - total arrival rate at scheduler $= \lambda_{ready} + \lambda_H + \lambda_C + \lambda_W$

μ_1; μ_2; μ_3 - unequal service rates of heterogeneous cores

- In a priority queue, the threads will be scheduled according to their priorities and in same priority class, they will be scheduled according to the First-Come-First-Service basis.
- As the cores are heterogeneous, the probability of the threads getting scheduled on a particular core is different for each core. Assume that, when the number of threads is less than the number of cores, threads will get scheduled by selecting the fastest core first. After that, the scheduler will take the thread scheduling decisions considering the core availability and core temperature variations.
- Arrival rates follow the law of conservation of the flows stated by Jackson[4].
- α, β, γ depend on cores' service rates and temperature variations of cores.

α - probability with which threads get scheduled on $core_1$

β - probability with which threads get scheduled on $core_2$

γ - probability with which threads get scheduled on $core_3$

λ_1, λ_2, λ_3 - arrivals rates at individual cores.

Figure 4.1 shows thread scheduling mechanism section with all the variables assigned:

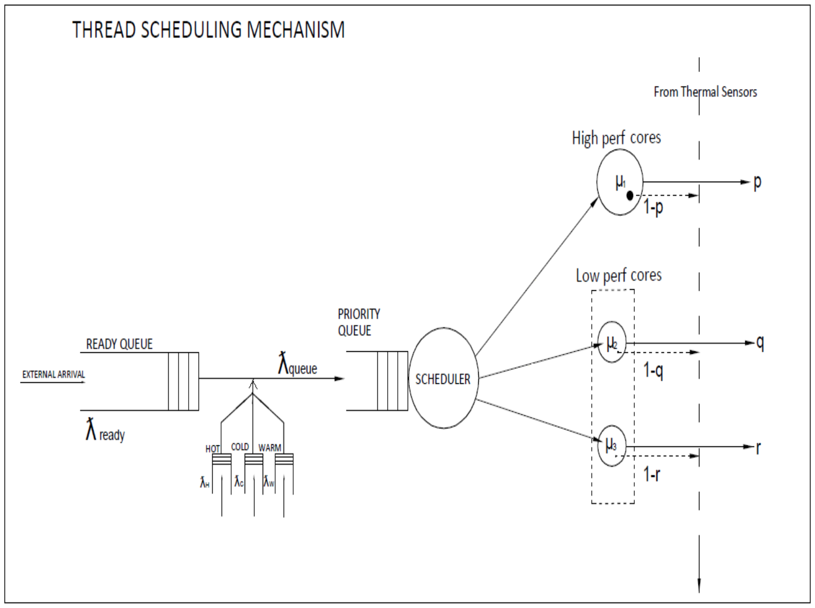


Figure 4.1: Thread scheduling mechanism

$\mu = \mu_1 + \mu_2 + \mu_3$

ρ - utilization, traffic intensity = λ/μ

p_0 - probability of 0 threads in the system

p_n - probability of n threads in the system

L_q - average number of threads in queue

W_s - average waiting time for heterogeneous cores

In the previous chapter, the equations for the average queue length and mean response time for the M/M/3 heterogeneous cores queuing model have been derived. In proposed model, threads are of two priorities: high and low and their scheduling will be preemptive in nature, where a high priority thread will preempt a low priority thread. Assume that,

$prob_{hpref}$ - probability of high priority threads

$prob_{lpref}$ - probability of low priority threads

λ_{hpref} - arrival rate for high priority threads = $prob_{hpref} * \lambda queue$

λ_{lpref} - arrival rate for low priority threads = $prob_{lpref} * \lambda queue$

where, $\lambda_{queue} = \lambda_{hpref} + \lambda_{lpref}$

$\rho_{hp} = \frac{\lambda_{hpref}}{\mu_1+\mu_2+\mu_3}$

$\rho_{queue} = \frac{\lambda_{queue}}{\mu_1+\mu_2+\mu_3}$

R_{high} - response time(average waiting time in heterogeneous cores) for high priority threads

R_{low} - response time(average waiting time in heterogeneous cores) for low priority threads

For the highest priority class, their won't be any preemption i.e., the highest priority threads are unaffected by the presence of threads of the lower priority. Hence, the mean response time for the high priority threads is same as equation 3.9, substituting λ and ρ values of high priority threads we get,

$$R_{high} = \frac{1}{\lambda_{high}} * \left[\sum_{n=1}^{2} \frac{n * \lambda_{high}^{n}}{\prod_{k=1}^{n} \sum_{j=1}^{k} \mu_j} + \frac{\lambda_{high}^{3}}{\prod_{k=1}^{3} \sum_{j=1}^{k} \mu_j} * \frac{3 - 3 * \rho_{hp} + \rho_{hp}}{(1-\rho_{hp})^2} \right] * p_{h0} \tag{4.1}$$

where, $p_{h0} = \left[1 + \sum_{n=1}^{2} \frac{\lambda_{high}^{n}}{\prod_{k=1}^{n} \sum_{j=1}^{k} \mu_j} + \frac{\lambda_{high}^{3}}{\prod_{k=1}^{3} \sum_{j=1}^{k} \mu_j * (1-\rho_{hp})} \right]^{-1}$

In preemptive thread scheduling to calculate response time for the low priority threads, equation stated below is valid for any system for which the high priority threads have finite waiting times [5],

$$R_{low} = \frac{(\lambda_{queue} * R_{queue} - \lambda_{high} * R_{high})}{\lambda_{low}} \tag{4.2}$$

where, R_{queue} - response time for all incoming threads,i.e., combined response time for high as well as low priority threads. λ_{queue} - combined arrival rate for high as well as low priority threads.

$R_{queue} = \frac{1}{\lambda_{queue}} * \left[\sum_{n=1}^{2} \frac{n*\lambda_{queue}^{n}}{\prod_{k=1}^{n} \sum_{j=1}^{k} \mu_j} + \frac{\lambda_{queue}^{3}}{\prod_{k=1}^{3} \sum_{j=1}^{k} \mu_j} * \frac{3-3*\rho_{queue}+\rho_{queue}}{(1-\rho_{queue})^2} \right] * p_{q0}$

where, $p_{q0} = \left[1 + \sum_{n=1}^{2} \frac{\lambda_{queue}^{n}}{\prod_{k=1}^{n} \sum_{j=1}^{k} \mu_j} + \frac{\lambda_{queue}^{3}}{\prod_{k=1}^{3} \sum_{j=1}^{k} \mu_j * (1-\rho_{queue})} \right]^{-1}$

Average queue length of priority queue is same as equation 3.10, where ρ is going to be ρ_{queue}, as priority queue is going to be common for both high as well as low priority threads.

$$L_{pri_q} = p_0 * \frac{(\mu_1 + \mu_2 + \mu_3)^3 * \rho_{queue}^4}{\prod_{k=1}^{3} \sum_{j=1}^{k} \mu_j} * \frac{1}{(1 - \rho_{queue})^2} \tag{4.3}$$

4.2 Thermal Aware Thread Classification

At departure, either threads will complete their tasks and they will leave the system or depending upon core temperature variations there may be a possibility that threads will be sent to the thermal profiler for rescheduling or low priority threads which are preempted by higher priority threads will come to thermal profiler. The thermal profiler's task is to classify the threads in hot/warm/cold queue. Assume, p be the probability of threads which will complete their tasks and they will exit the system from core 1 then, $(1-p)$ will be the probability that threads will go to thermal profiler from core 1. Similarly, q be the probability of threads completing tasks and $(1-q)$ will be the probability that threads will go to the thermal profiler from core 2. Finally, r be the probability of threads which will complete their tasks and they will exit the system from core 3 so, $(1-r)$ will be the probability that threads will go to the thermal profiler.

Probability of threads getting rescheduled(going to thermal profiler) depends on :

- Preemption due to the high priority threads
- Preemption of the threads as the core temperature increases beyond high threshold of the core

Assume that,

$\lambda_{profiler}$ - arrival rate at thermal profiler

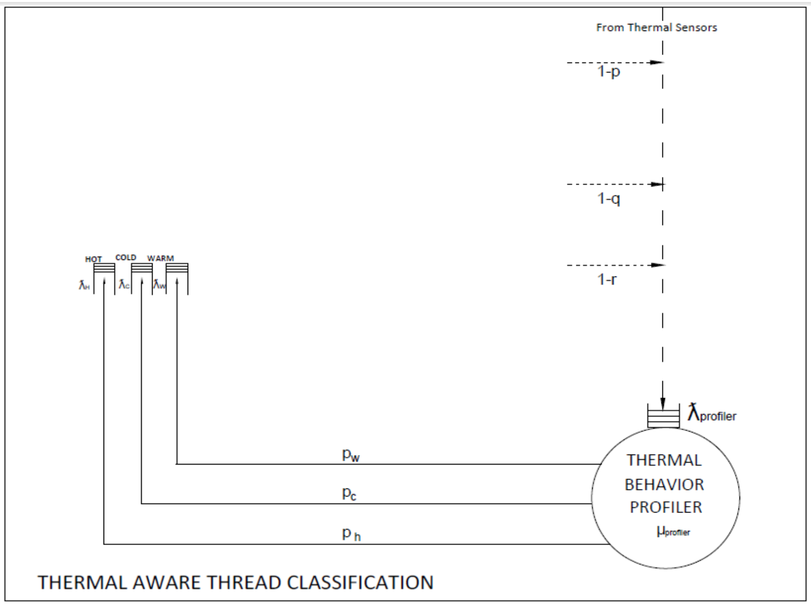


Figure 4.2: Thermal aware thread classification model

$\mu_{profiler}$ - service rate of thermal profiler

$R_{profiler}$ - response time for thermal profiler

L_{prof_q} - average length of profiler queue

$\rho_{profiler}$ - traffic intensity at profiler $= \frac{\lambda_{profiler}}{\mu_{profiler}}$

α, β, γ are probabilities with which threads arrive at core 1, core 2 and core 3 respectively and λ_{queue} is arrival rate at priority queue.

Hence, by Jackson's theorem,

$\lambda_{profiler} = ((1-p) * \alpha + (1-q) * \beta + (1-r) * \gamma) * \lambda_{queue}$

The thermal profiler's task is just to classify threads, hence it will act as per the M/M/1 queuing model with a first-come-first-serve queue. From equations 3.2 and 3.3,

$$R_{profiler} = \frac{1}{\mu_{profiler} - \lambda_{profiler}} \tag{4.4}$$

$$L_{prof_q} = \frac{\rho^2_{profiler}}{1 - \rho_{profiler}} \tag{4.5}$$

Thermal profiler will classify all threads into three queues, hot, warm and cold. Assume that,

p_h - probability of threads going to hot queue

p_w - probability of threads going to warm queue

p_c - probability of threads going to cold queue

From the above mentioned probabilities we can evaluate the arrival rate the thermal profiler queue which gives the distribution of arrival rates in the hot, warm and cold queue.

λ_h - arrival rate for hot queue $= \lambda_{profiler} * p_h$

λ_w - arrival rate for warm queue $= \lambda_{profiler} * p_w$

λ_c - arrival rate for cold queue $= \lambda_{profiler} * p_c$

4.3 Calculation of the Arrival Rate at the Priority Queue

As per Jackson's theorem, using the external arrival rates and the probability distribution of all the arrival rates, we can calculate the arrival rate at any queue in the queuing network [4]. Same logic can be implemented in the proposed queuing network to calculate the arrival rate at the priority queue(λ_{queue}) in terms of the external arrival rate(λ_{ready})

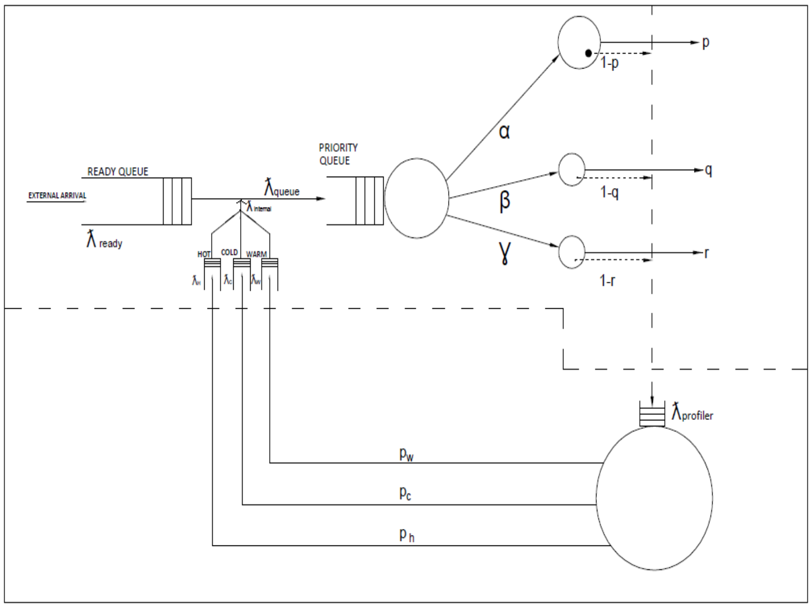


Figure 4.3: Queuing model with probability distributions and arrival rates

From figure 4.3, we can combine arrival rates of hot, warm and cold queues into single arrival rate.

$\lambda_{internal} = \lambda_{hot} + \lambda_{warm} + \lambda_{cold}$

Also, we get one more relation,

$\lambda_{internal} = \lambda_{profiler}$

From previous section, we know value of $\lambda_{profiler}$, hence,

$\lambda_{internal} = ((1-p)*\alpha + (1-q)*\beta + (1-r)*\gamma)*\lambda_{queue}$

At input, we have,

$\lambda_{queue} = \lambda_{internal} + \lambda_{ready}$

$\lambda_{queue} - ((1-p)*\alpha + (1-q)*\beta + (1-r)*\gamma)*\lambda_{queue} = \lambda_{ready}$

$\lambda_{queue} = \frac{\lambda_{r}eady}{1-((1-p)*\alpha+(1-q)*\beta+(1-r)*\gamma)}$

4.4 Analysis of Results

Equations 4.1, 4.2, 4.3, 4.4 and 4.5 give all the values that we need to determine average response time of system and average queue lengths. Assume that,
$Response_{system_{high}}$ - Response time of system for high priority threads
$Response_{system_{low}}$ - Response time of system for low priority threads
$QueueLength_{priority}$ - Average queue length of priority queue
$QueueLength_{profiler}$ - Average queue length of profiler queue
Response time of system for high priority threads = Average waiting time in heterogeneous cores for high priority threads + Average waiting time in thermal profiler

Hence, by adding equations 4.1 and 4.4, we get, response time of system for high priority queues :

$$Response_{system_{high}} = R_{high} + R_{profiler} \tag{4.6}$$

Response time of system for low priority threads = Average waiting time in heterogeneous cores for low priority threads + Average waiting time in thermal profiler

Hence, by adding equations 4.2 and 4.4, we get, response time of system for low priority queues :

$$Response_{system_{low}} = R_{low} + R_{profiler} \tag{4.7}$$

By equation 4.3 we get, average queue length of priority queue :

$$QueueLength_{priority} = L_{pri_q} \tag{4.8}$$

By equation 4.5 we get, average queue length of profiler queue :

$$QueueLength_{profiler} = L_{prof_q} \tag{4.9}$$

Equations 4.6, 4.7, 4.8 and 4.9 give average response times and average queue lengths, by using these equations for different set of input values we will do Performance Evaluation of our system.

In the proposed model, the external arrival rate λ_{ready} is variable, the probability distributions of the threads $(\alpha, \beta, \gamma, p, q, r, prob_{hpref}, prob_{lpref})$ and the service rates $(\mu_1, \mu_2, \mu_3, \mu_{profiler})$ are kept constant.

For all the cases, assume that,

$\alpha = 0.5$; $\beta = 0.3$; $\gamma = 0.2$;

$prob_{hpref} = 0.6$; $prob_{lpref} = 0.4$;

$p = 0.7$; $q = 0.6$; $r = 0.5$

4.4.1 Case 1 : Variable Arrival Rates

In this case, the system's response times and queue lengths are calculated for different values of external arrival rates. Assume that the service rates are constant for all the cores and the thermal profiler.

$\mu_1 = 250$; $\mu_2 = 200$; $\mu_3 = 150$

$\mu_{profiler} = 300$

Table 4.1: Response times and queue lengths for variable arrival rates

External Arrival Rate	Average Response time for high priority	Average Response Time for Low Priority	Priority Queue Length	Profiler Queue Length
100	0.0083	0.0088	0.0149	0.0477
150	0.0091	0.0102	0.0782	0.1221
200	0.01	0.0123	0.2711	0.2519
250	0.0113	0.0162	0.7945	0.4691
300	0.0132	0.0248	2.3528	0.8358
350	0.0162	0.0624	10.6477	1.4913
375	0.0185	0.5325	122.9603	2.0271

Plot the graphs of the response times and queue lengths versus external arrival rates for the values shown in table 4.1.

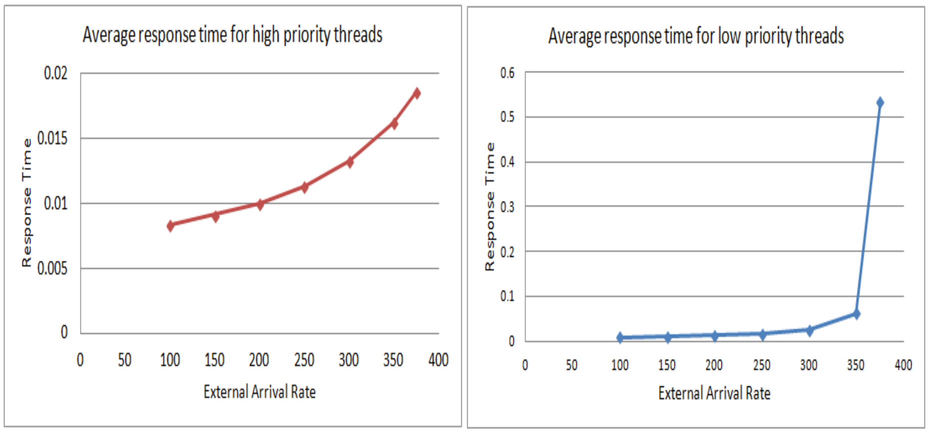
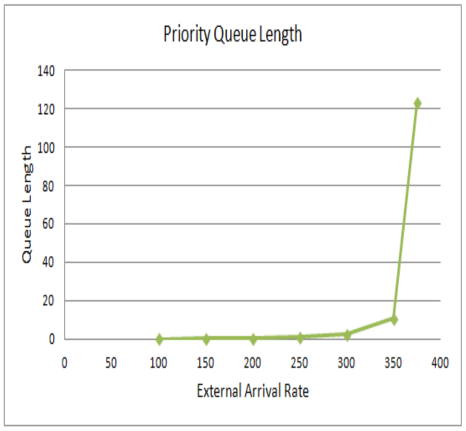


Figure 4.4: Response time graphs for variable arrival rates



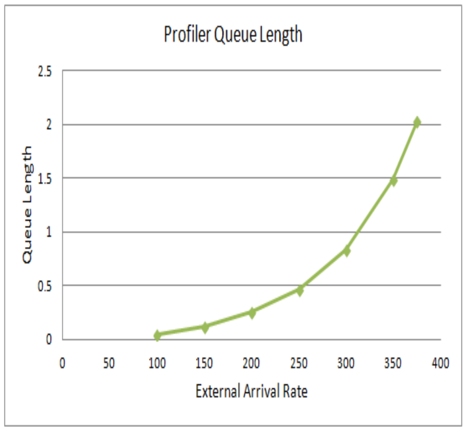


Figure 4.5: Queue length graphs for variable arrival rates

For all the above derivations of response times and queuing lengths, the system is considered to be in a steady state. Therefore, ρ i.e., traffic intensity must be less than 1.

In general notation, $\rho = \frac{\lambda}{\mu}$. Hence, for the system to be in a steady state, the arrival rates must be less than the service rates. In a steady state, the response times and queue lengths are finite for any system.

From graphs 4.4 and 4.5, as the arrival rates approach the total service rate of the system, the response times and queue lengths start increasing exponentially. Incoming threads at priority queue are scheduled on the heterogeneous cores. Hence, care must be taken that the arrival rate of the priority queue must be less than the service rates of heterogeneous cores i.e., $\frac{\lambda_{queue}}{\mu_1+\mu_2+\mu_3} < 1$. Similarly, the arrival rate at the profiler queue must be less than the service rate of the thermal profiler i.e., $\frac{\lambda_{profiler}}{\mu_{profiler}} < 1$.

Due to preemptive scheduling, the response time for the low priority threads is higher than the response time for the high priority threads.

4.4.2 Case 2 : Fastest Core First (FCF) v/s Slowest Core First(SCF)

In this case, two different systems are analyzed. In the proposed method, the first arriving thread will get scheduled to the fastest core, the second thread to the second fastest core and when all the cores are busy, the scheduler will take decisions. This proposed model is compared with the slowest core first model, where the first thread goes to the slowest core. Comparison of the response times and average queue lengths for both the cases will show which method is better. Arrival rate and probability distribution values are kept same as that of the case 1, service rates for case 2 are:

$\mu_1 = 150$; $\mu_2 = 200$; $\mu_3 = 250$

$\mu_{profiler} = 300$

Table 4.2: Response times and queue lengths for SCF policy

External Arrival Rate	Average Response time for high priority	Average Response Time for Low Priority	Priority Queue Length	Profiler Queue Length
100	0.0103	0.01	0.0229	0.0477
150	0.0108	0.0108	0.1068	0.1221
200	0.0115	0.0127	0.3369	0.2519
250	0.0126	0.0164	0.9151	0.4691
300	0.0144	0.0249	2.5449	0.8358
350	0.0172	0.0624	10.9265	1.4913
375	0.0195	0.5325	123.2876	2.0271

Comparing results of tables 4.1 and 4.2 using graphical representation,

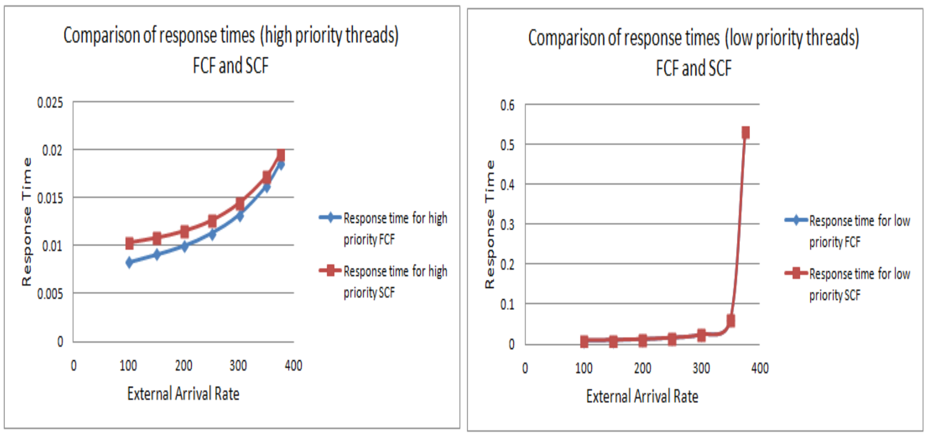


Figure 4.6: Comparison of response times for FCF and SCF models

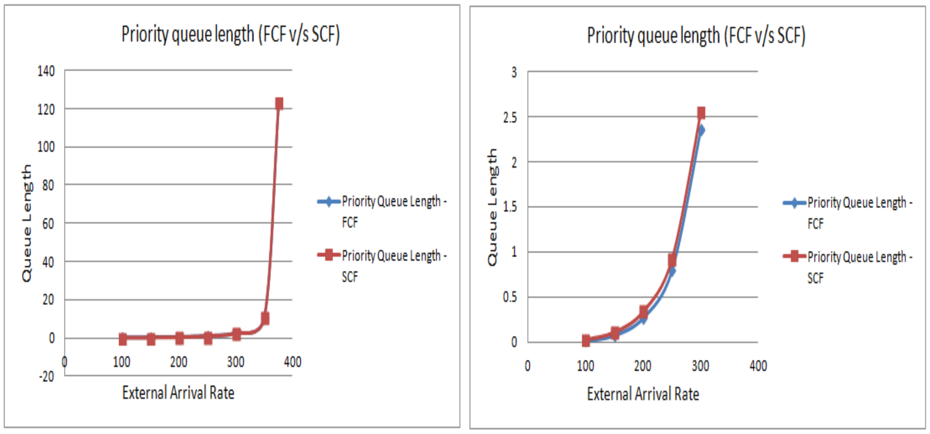


Figure 4.7: Comparison of queue lengths for FCF and SCF models

The profiler queue length is going to be the same in both the cases, as the service rate of the profiler and the probability distribution of the threads are kept constant. Hence, only the response times and priority queue length for FCF and SCF models are compared.

From the graphs 4.6 and 4.7, the response time for high priority threads is better in the fastest core first (FCF) method. For the low priority threads, the response times are approximately the same for both the models. For smaller values of the arrival rates, the queue length for FCF approach is less than the SCF approach. These results show that the fastest core first (FCF) method is better than the slowest core first (SCF) method for thread scheduling in heterogeneous multi-core architectures.

4.4.3 Case 3 : Heterogeneous Cores v/s Homogeneous Cores

In this case, two different configurations of the cores are compared. The proposed heterogeneous core model is compared with the homogeneous core model. For homogeneous core model, the service rate will be the same for all the three cores. If the service rates for the homogeneous cores is same as that of the service rate of the fastest core of our model, better response times and minimum queue lengths can be obtained.

But, the basic idea of using heterogeneous cores in place of homogeneous cores is that different threads have different resource requirements during their execution. If threads are assigned depending upon their performance requirements, full utilization of all the available resources can be avoided. This will help in achieving performance enhancement and improvement in power consumption of the cores.

For the proposed model, assume that, $\mu_1 = 250$; $\mu_2 = 200$; $\mu_3 = 150$. Hence, the average arrival rate is at around 200. Consider a homogeneous system with an average service rate of 200. Table 4.3 shows the response times and queue lengths for the homogeneous core model:

Table 4.3: Response times and queue lengths for homogeneous cores

External Arrival Rate	Average Response time for high priority	Average Response Time for Low Priority	Priority Queue Length	Profiler Queue Length
100	0.0093	0.0096	0.0222	0.0635
150	0.0101	0.011	0.1111	0.1667
200	0.0112	0.0135	0.3747	0.3556
250	0.0129	0.0184	1.1005	0.6944
300	0.0158	0.0314	3.5112	1.3333
350	0.0213	0.1602	33.1791	2.7222

Comparing results of tables 4.1 and 4.3 using graphical representation,

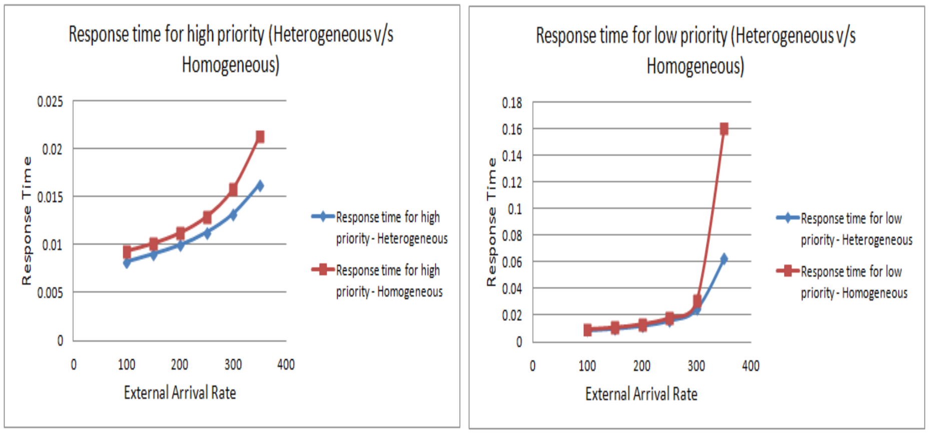


Figure 4.8: Comaprison of response times for homogeneous v/s heterogneous cores

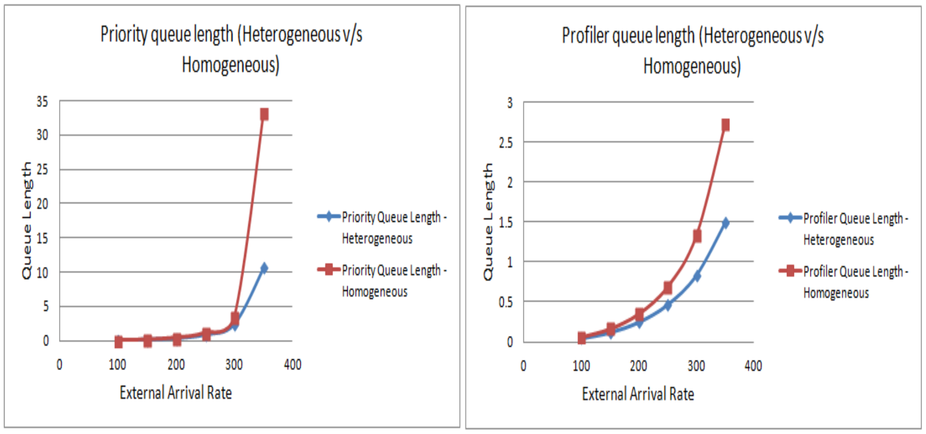


Figure 4.9: Comaprison of queue lengths for homogeneous v/s heterogneous cores

From the graphs and 4.8 and 4.9, the heterogeneous multi-core architectures with FCF policy show better performance than the homogeneous multi-core architectures in some cases. It can be seen that, there is a steep growth in the response times and service rates for heterogeneous multi-core architectures as arrival rates approach service rates.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

In the presented work, a core temperature aware, online estimation based thread scheduling algorithm for heterogeneous multi-core architectures was proposed. The key idea is to determine the thread's behavior on all the cores and to select a thread-to-core match which would help in achieving the improved performance and power efficiency of the cores. While doing so, the scheduler will take the thread's resource requirement and the individual core temperature into the consideration. The proposed design is analyzed using queuing theory models.

The fastest core first policy for thread scheduling on the heterogeneous cores show better results over the slowest core first policy. For any system to be stable, the arrival rate of the threads must be less than the service rates of the cores i.e., the traffic intensity must be less than 1. Heterogeneous cores without a thermal management unit can show much better response times and average queue lengths, but they are prone to performance degradations due to hot spots and steep temperature gradients of the cores. These type of cores without any thermal protection techniques have a high probability of getting damaged.

To improve the power efficiency and performance of the system, a suitable thread-to-core pair must be selected using online estimation. Using this method for thread scheduling, we can avoid sampling overheads associated with an online sampling method. Also, there is no need to have prior knowledge about the thread's behavior.

Temperature-aware scheduling with the classification of threads into hot/warm/cold threads is done dynamically at run time, which makes the prediction more accurate. By using a core-temperature aware scheduling, core temperatures can be prevented from getting into the critical region. This will increase the overall lifespan of the system and the power consumption efficiency can be improved. The proposed DTM technique prevents thermal problems before they occur, as opposed to reacting after hot spots.

REFERENCES

[1] Zhuravlev, S.; Saez, J.C.; Blagodurov, S.; Fedorova, A.; Prieto, M., 2013, "Survey of Energy-Cognizant Scheduling Techniques," *Parallel and Distributed Systems, IEEE Transactions on* , vol.24, no.7, pp.1447-1464.

[2] Cheng-Yu Lee; Shuang-Jhu Yang; Rong-Guey Chang, 2013, "Thermal-Aware Scheduling Collaborating with OS and Architecture," *Parallel Processing (ICPP), 2013 42nd International Conference on*, pp.1044-1051.

[3] Rodrigues, R.; Annamalai, A.; Koren, I.; Kundu, S., 2012, "Scalable Thread Scheduling in Asymmetric Multicores for Power Efficiency," *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, pp.59-66.

[4] Virtoma, J., 2005, "Queueing Theory, 38.3143," Lecture Notes.

[5] Buzen, J. P.; Bondi, A. B., 1983, "The response times of priority classes under preemptive resume in M/M/m queues," *Operations Research 31*, no. 3, pp.456-465.

[6] Held, J.; Bautista J.; Koehl S., 2006, "White paper from a few cores to many: A tera-scale computing research review".

[7] Khan, O.; Kundu, S., 2011, "Microvisor: a runtime architecture for thermal management in chip multiprocessors," *In Transactions on High-Performance Embedded Architectures and Compilers IV*, pp. 84-110. Springer Berlin Heidelberg.

[8] Brooks, D.; Martonosi, M, 2001, "Dynamic thermal management for high-performance microprocessors," *In High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pp. 171-182.

[9] Lee, S.; Gaudiot, J, 2006, "Throttling-based resource management in high performance multithreaded architectures," *Computers, IEEE Transactions on*, vol. 55, no. 9, pp.1142-1152.

[10] Skadron, K.; Stan, M.R.; Wei Huang; Velusamy, S.; Sankaranarayanan, K.; Tarjan, D., 2003, "Temperature-aware microarchitecture," *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pp.2-13.

[11] Buzen, J. P., 1971, "Queueing network models of multiprogramming," *Harvard university, division of engineering and applied physics.*

[12] Khan, O.; Kundu, S., 2010, "A self-adaptive scheduler for asymmetric multicores," *In Proceedings of the 20th symposium on Great lakes symposium on VLSI*, pp.397-400.

[13] Shelepov, D.; Alcaide, J. C.; Jeffery, S.; Fedorova, A.; Perez, N.; Huang, Z.; Blagodurov; Kumar, V, 2009, "HASS: a scheduler for heterogeneous multicore systems." *ACM SIGOPS Operating Systems*, Review 43, no. 2, pp.66-75.

[14] Sherwood, T.; Sair, S.; Calder, B., 2003, "Phase tracking and prediction," *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pp.336-347.

[15] Kumar, R.; Farkas, K.I.; Jouppi, N.P.; Ranganathan, P.; Tullsen, D.M., 2003, "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction," *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pp.81-92.

[16] Becchi, M.; Crowley, P., 2006, "Dynamic thread assignment on heterogeneous multiprocessor architectures." *In Proceedings of the 3rd conference on Computing frontiers*, pp.29-40.

[17] Srinivasan, S.; Zhao, L.; Illikkal, R.; Iyer, R, 2011 "Efficient interaction between OS and architecture in heterogeneous platforms." *ACM SIGOPS Operating Systems*, Review 45, no. 1, pp.62-72.

[18] Rodrigues, R.; Annamalai, A.; Koren, I.; Kundu, S.; Khan, O., 2011, "Performance Per Watt Benefits of Dynamic Core Morphing in Asymmetric Multicores," *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pp.121-130.

[19] Kumar, A; Li Shang; Li-Shiuan Peh; Jha, N.K., 2006, "HybDTM: a coordinated hardware-software approach for dynamic thermal management," *Design Automation Conference, 2006 43rd ACM/IEEE*, pp.548-553.

[20] Yang Ge; Malani, P.; Qinru Qiu, 2010, "Distributed task migration for thermal management in many-core systems," *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pp.579-584.

[21] Jeonghwan Choi; Chen-Yong Cher; Franke, H.; Hamann, H.; Weger, A.; Bose, P., 2007, "Thermal-aware task scheduling at the system software level," *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pp.213-218.

[22] Coskun, A.K.; Rosing, T.S.; Whisnant, K., 2007, "Temperature Aware Task Scheduling in MPSoCs," *Design, Automation and Test in Europe Conference and Exhibition*, pp.1-6.

[23] Lazowska, E. D.; Zahorjan, J.; Graham, G.; Sevcik, K. C., 1984, *Quantitative system performance: computer system analysis using queueing network models*, Prentice-Hall, Inc.

[24] Jian Chen; John, L.K., 2009, "Efficient program scheduling for heterogeneous multi-core processors," *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pp.927-930.

[25] Gross, D.; Shortle, J. F.; Thompson, J.; Harris, C., 2013, *Fundamentals of queueing theory*, John Wiley and Sons.

[26] Kleinrock, Leonard, 1975, *Queueing systems, volume I: theory.*

[27] Alves, F. S. Q.; Yehia, H. C.; Pedrosa, L. A. C.; Cruz, F. R. B.; Laoucine, K., 2011, "Upper Bounds on Performance Measures of Heterogeneous M/M/c Queues." *Mathematical Problems in Engineering.*

[28] Wang, T.; Jau-Chuan Ke; Kuo-Hsiung Wang; Siu-Chuen Ho, 2006, "Maximum likelihood estimates and confidence intervals of an M/M/R queue with heterogeneous servers." *Mathematical Methods of Operations Research 63*, no. 2, pp.371-384.

[29] Winter, J. A.; Albonesi D. H.; Shoemaker C. A., 2010, "Scalable thread scheduling and global power management for heterogeneous many-core architectures." *In Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pp. 29-40.