Energy Efficient Multicore Scheduling Algorithms for Real Time Systems

THESIS

Submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by
MAYURI A. DIGALWAR

Under the Supervision of
Prof. Sudeept Mohan
and
Prof. Biju K. Raveendran

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
2016

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI (Rajasthan)

CERTIFICATE

This is to certify that the thesis entitled "**Energy Efficient Multicore Scheduling Algorithms for Real Time Systems**" and submitted by Mrs. Mayuri A. Digalwar, ID No. 2009PHXF432P, for award of Ph.D. degree of the institute embodies original work done by her under our supervision.

Signature of Co-Supervisor	Signature of Supervisor
Name: BIJU K. RAVEENDRAN	Name: SUDEEPT MOHAN
Designation: Assistant Professor	Designation: Professor
Department of CSIS,	Department of CSIS,
BITS, Pilani, Goa Campus	BITS, Pilani, Pilani Campus
Date:	Date:

Dedicated to

My Family

Acknowledgement

My journey of Ph.D. required the countless and selfless support, generosity and time of many people in my personal and academic life. I am glad to take the opportunity to acknowledge and thank all of them.

I am deeply grateful to Prof. Sudeept Mohan, my supervisor, for his constant support, guidance and kindness during my pursuit in Ph.D. degree. This work would not have been possible without his guidance and involvement, his support and encouragement on daily basis from the beginning till this moment. His zeal of perfection and commitment towards work always inspired me to do more. He has always been a great advisor, teacher and mentor for me in my research as well as in my academic career. I am thankful to him for having long discussions with me and giving me invaluable suggestions which helped me to grow my understanding and gain maturity in research. I particularly appreciate him for finding time from his busy schedule whenever I asked for it. I deeply express my gratitude and thank him for his support and concern.

While many students are fortunate to have a single supervisor, I have been blessed with two. I am deeply grateful to Prof. Biju K. Raveendran, my co-supervisor, for his invaluable guidance and support throughout my Ph.D. work. He has been a constant source of encouragement to me. His expertise and experience further enriched my knowledge and understanding in research. The discussions with him have always shown me new directions and made me curious to go into the depth of concept. Since he is located in BITS Pilani, Goa campus, we use to interact with each other through skype calls and emails. I am thankful to him for making this research work successful under this constraint of location.

I am thankful to Prof. Souvik Bhattacharyya, Vice Chancellor, BITS Pilani and Prof. Ashoke Kumar Sarkar, Director, BITS Pilani, Pilani campus for giving me an opportunity to pursue my Ph.D. research with financial support. I am thankful to Prof. Sanjay Kumar Verma, Dean, and Prof. Hemant R. Jadhav, Associate Dean, Academic Research Division (Ph.D. Programme) for providing necessary guidelines and extending full support, which were very important for the successful completion of this thesis. I would like to thank Prof. S.C. Sivasubramanian, Dean, Administration and Prof. J.P. Misra, Unit Chief, Information

Processing Centre for their encouragement and support during the entire duration of this research.

Besides my advisors, I would like to thank to my Doctoral Advisory Committee (DAC) members, Prof. S. Gurunarayanan, Dean, WILP and Dr. Abhishek Mishra, Assistant Professor in Computer Science and Information Systems for evaluating my work and giving me valuable comments and suggestions. I also thank all the reviewers of various conferences and journals for reviewing our research papers and providing their valuable comments and suggestions for improvement.

I am thankful to the Head of the Department, Prof. Rahul Banerjee for all his support in granting funds for attending conferences and his constant encouragement throughout. I would like to thank the Convener and all members of Doctoral Research Committee (DRC) for their constant support and guidance. My sincere thanks also go to Prof. Sundar Balasubramaniam, Prof. Navneet Goyal, Prof. Poonam Goyal and Dr. Virendra Singh Shekhawat. I am grateful to them for giving me valuable feedback and suggestions during the departmental seminars which had strengthen my work over time. I am also thankful to Mr. Sanwarmal for his administrative support throughout.

I thank my fellow colleagues who have been a constant source of motivation, encouragement and inspiration. Particularly, my friends, Vandana, Avinash and Asma were always been there to support me technically through stimulating discussions. I had received a big support from them in difficult situations during my research work. I would also like to acknowledge one of my past post-graduate students, Mr. Pravin Gahukar with whom I have worked on one of the problems in this thesis.

A special thanks to my family. Words cannot express how grateful I am to my mother-in-law, Sau. Ratna Digalwar and father-in-law, Shri. K. M. Digalwar for the sacrifices made by them on my behalf. Without their constant support and motivation right from the beginning till this moment, I would not have finished my work peacefully. I deeply owe to my parents, who motivated and helped me at every stage of my life. I miss a lot my father, Late. Shri. Nishikant Rajurwar who is not with me to share this moment of happiness. I would like to show my gratitude towards my mother, Smt. Maya Rajurwar for her love and trust which gave me strength throughout my life and in this Ph.D. work.

I owe thanks to a special person, my husband, Prof. Abhijeet Digalwar for his continued and unfailing love, support and understanding during my pursuit of Ph.D. degree that made completion of this thesis possible. He was always there with me in the moments when there was no one to solve my difficulties. I greatly value his contribution and deeply appreciate his belief in me. Last but not the least, I appreciate my son, Akshat for the long lasting patience and understanding he showed during the entire Ph.D. work and thesis writing. I consider myself luckiest to have such a loving and caring son.

Lastly, I thank the Almighty for giving me strength and patience to work.

Mayuri

Abstract

With the advancement of technology and ever increasing demand of portable, scalable and sophisticated embedded systems, managing energy consumption to prolong the battery life of embedded devices has become a big challenge. With the advent of multi-core processors in the embedded market, reducing the energy consumption is becoming increasingly important for multi-core processors as well.

Modern multi-core processors consume two types of energy, viz., dynamic and static energy. Dynamic energy is consumed due to switching activity whereas static energy is consumed due to increase in leakage current. These processors have capability to dynamically lower the supply voltage that reduces dynamic energy consumption. However, reducing supply voltage increases gate delay which requires one to lower operating frequency. As a consequence, the tasks take more time to execute. In this thesis, we have focused on real time embedded systems that execute hard and soft real time tasks. The major challenge for these systems is to optimize energy consumption using dynamic voltage and frequency scaling (DVFS) without missing the timing constraints of the hard real time tasks and responsiveness of the soft real time tasks. The energy saving achieved by DVFS is severely limited with the dramatic increase in leakage power consumption. Therefore, to minimize the overall energy consumption, there is a need to optimize dynamic as well as static energy consumption.

This thesis addresses the issue of overall energy optimization in real time embedded systems at the operating system level using efficient real time task scheduling algorithms. The proposed energy efficient scheduling algorithms, Energy Efficient Dynamic Voltage and Frequency Scaling (EEDVFS) and Energy Efficient Uni-Core Scheduler (EE-UCS) optimize dynamic energy consumption of uniprocessor. Another proposed energy efficient scheduling algorithm, Multi-Core Scheduler (MCS) optimizes dynamic energy consumption of homogeneous multi-core processors. These algorithms are capable of scheduling the hard and soft real time tasks together. They use dynamic voltage and frequency scaling technique to reduce dynamic energy consumption. The slack reclamation scheme devised to select the optimal frequency is very aggressive and achieves maximum energy saving. At the same time, the method of allocation and scheduling of soft real time tasks helps to achieve acceptable response time. But the limitation of these algorithms is that they are not capable of reducing static energy consumption. Therefore, we proposed an energy efficient scheduling algorithm - Leakage Aware Multi-Core Scheduler (LAMCS), which is an extension to the previous algorithm MCS. The proposed LAMCS algorithm is capable of minimizing both dynamic and static energy consumption resulting in overall energy minimization. LAMCS is also capable of scheduling hard and soft real time tasks together. Along with DVFS technique, LAMCS uses dynamic shutdown and procrastination schemes to reduce dynamic as well as static energy consumption.

A full-fledged simulation tool has been developed as a significant part of this work in order to implement, test and evaluate the performance of the proposed algorithms. The simulation tool is divided into two major categories - Task Scheduling and Task Set Generation. Current simulator includes implementations of Earliest Deadline First (EDF), EDF with Total Bandwidth Server (TBS) and Deferrable Server (DS), cycle conserving EDF with TBS and DS for uniprocessor and multi-core processor platforms, DVFS based multi-core scheduler implementation for mixed work load (MCS), leakage aware scheduler, DVFS based leakage aware multi-core scheduler (LAMCS) etc. It has modules to generate synthetic task sets of two types: periodic task sets and mixed task sets. The mixed task sets contain hard and soft real time tasks. There are other modules which are responsible for the calculation of various performance metrics such as energy consumption, aperiodic task's response times, various decision counts such as scheduling points, preemption count, migration count, cache impact points etc.

The simulation tool is written in java programming language that makes use of object oriented paradigm. The graphical User Interface (GUI) of simulator is very user friendly and is easy to explore and use. The use of abstract classes facilitates addition of new scheduling algorithms in the current version of simulator. Finally, an important and novel aspect of simulator is its ability to produce analytical results of the algorithms by plotting various graphs.

The proposed algorithms in this work are extensively tested and evaluated using synthetically generated benchmark suites. The parameters of energy consumption used in all the experiments are taken from the Transmetta Crusoe processor.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

DVFS	Dynamic Voltage and Frequency Scaling
DPM	Dynamic Power Management
wcet	worst case execution time
aet	actual execution time
CMOS	Complementary Metal Oxide Semiconductor
EDF	Earliest Deadline First
SNTA	Stretching to Next Task Arrival
PBSS	Priority Based Slack Stealing
UU	Utilization Updating
ccEDF	Cycle conserving Earliest Deadline First
laEDF	Look ahead Earliest Deadline First
DS	Deferrable Server
SS	Sporadic Server
CBS	Constant Bandwidth Server
TBS	Total Bandwidth Server
SNRT	Stretching to Next Replenishment Time
BBSS	Bandwidth Based Slack Stealing
POSD	Periodic Only Slack Distribution
WSE	Workload based Slack Estimation
BRS	Basic Reclamation Scheme
MRS	Mutual Reclamation Scheme
BSS	Bandwidth Sharing Scheme
RARA	Ratio based Aggressive Reclaim Algorithm
EEDVFS	Energy Efficient Dynamic Voltage and Frequency Scaling
GRUB – PA	Greedy Reclamation of Unused Bandwidth - Power Aware
DFSA	Deadline Based Frequency Scaling Algorithm
LC-EDF	Leakage Controlled Earliest Deadline First
OWAA	Optimal Workload Aware Algorithm
WFD	Worst Fit Decreasing

FFD	First Fit Decreasing
BFD	Best Fit Decreasing
AMBFF	Adaptive Minimal Bound First Fit
HeLP	Hetero Efficiency to Logical Processor
HeLP-TM	Hetero Efficiency to Logical Processor - Temporal Migration
GMF	Growing Minimum Frequency
LTF	Largest Task First
MCS	Multi-Core Scheduler
LAMCS	Leakage Aware Multi-Core Scheduler
SVFS	Static Voltage and Frequency Scaling
Non-DVFS	Non- Dynamic Voltage and Frequency Scaling
STREAM	Simulation Tool for Real time Energy efficient scheduling and Analysis for Multi-core processors
PLL	Phase Locked Loops
LLREF	Largest Local Remaining Execution time First

Chapter 1

Introduction

This chapter provides an introduction to the research work presented in this thesis. It explains the motivation for pursuing this work and describes the research background. In addition, it introduces the research work carried out in this thesis and finally, it presents the organization of the thesis.

1.1 Motivation

With the rapid growth in technology, the contemporary computing systems available in today's era are shrinking in size and weight, exhibiting high performance and are capable of communicating with each other over the network. This has made embedded systems common place in everyday life. Unlike general purpose systems, embedded systems receive input from different sources through sensors and provide output to different devices through actuators without human intervention. These systems are used in many diverse application areas namely, automated industry applications, automotive applications, avionics, defense applications, consumer electronics etc. Many of the embedded systems are specially made for performing real time tasks where the timing constraints are important. Such systems are known as real time embedded systems. For example, in a missile guided system, the highly critical hard real time tasks like target sensing and track correction require an independent system mounted on the missile to sense the target and correct the path of the missile. If these tasks are not completed in time, the missile may home onto unwanted area and cause disaster (Mall, 2010). The systems which are designed to run such critical applications need powerful processors which are capable of performing intensive computations. These powerful processors consume significant amount of energy. Majority of these real-time embedded systems operate on battery. Therefore, the key design issues of real time embedded systems are energy efficiency and code density as these systems are expected to perform complex functionalities within limited power budget and small memory foot print. In addition, the modern real time embedded systems run applications that are dynamic and interactive in nature in which it is required to take input from the user while executing time constrained tasks. These interactive tasks, also known as aperiodic tasks, arrive arbitrarily in time and need quick response for good performance. Therefore, the responsiveness of the aperiodic tasks is also an important concern (Shin and Kim, 2006; Brandenburg and Anderson, 2007; Kato and Yamasaki, 2008).

Embedded systems are made up of one or more micro-processors / micro-controllers that are connected via interconnection network. The power supply is non-uniformly distributed over various components of the system which leads to variable power density. The components that are used frequently and do intensive computations consume more power than other components. The areas with more power density generate more heat and result in increase in temperature and may lead to system failure (Tiwari et al., 1996). In addition, heat dissipation becomes more challenging in embedded systems as compared to general purpose systems due to their small size. Therefore, energy optimization is an important issue in order to get longer battery life as well as for keeping the system free from failures.

Majority of the real time embedded systems now-a-days make use of sophisticated applications which require complex software and hardware. This raises the need of powerful processor design. In order to design such processors, the designers are not focusing on miniaturization of single processor since this leads to greater energy consumption and excessive heat dissipation. Instead, there is an increasing trend towards multi-core / multi-processor systems for real time embedded applications (Davis and Burns, 2011).

Realizing the ever increasing demand of high performance multi-core processors in battery operated real time embedded systems, many researchers have concentrated on the energy efficiency of these systems (Yang et al., 2005; Seo et al., 2008; Devdas and Aydin, 2010; Lu and Guo, 2011; Khandhalu et al., 2011; He and Muller, 2012a; Zhao et al., 2013). Efforts have been made to minimize the processor energy consumption at various levels such as architecture level, operating system level, compiler level, application and system program level etc (Saha and Ravindran, 2012). Many solutions have been proposed by hardware and software designers to deal with the problem of energy optimization in embedded systems and researchers are further working in this area.

This thesis addresses the issue of energy consumption of multi-core processor based real time embedded systems at the operating system level with the help of real time task scheduling and various energy optimization techniques. In this thesis, various energy efficient task scheduling algorithms are proposed for the optimization of both dynamic and static energy consumptions on uniprocessor and multi-core platforms. Realizing the importance of aperiodic tasks, all the proposed scheduling algorithms are capable of scheduling mixed task sets containing a mix of periodic and aperiodic tasks.

1.2 Research Background

The work presented in this thesis concentrates on the issue of energy consumption in multi-core processor based real time embedded systems.

1.2.1 Processor Energy Consumption

Viredaz and Wallach (2003) have stated that the processor cores consume majority of the energy as compared to other hardware components. The two main components of CMOS processor level energy consumption are static energy component due to leakage current and dynamic energy component due to switching activities (Duarte et al., 2002). There exist various strategies for reducing dynamic energy consumption like clock gating, power gating, transistor sizing, low power logic synthesis, DVFS etc (Benini et al., 1994; Tiwari et al., 1996; Borah et al., 1996; Macii et al., 2008; Li et al., 2011; Kim et al., 2002; Raja et al., 2006; Roy et al., 2003). Most of the modern processors in modern embedded systems are equipped with various levels of discrete voltages and frequencies which allow the execution of tasks at different voltages and frequencies (Burd and Brodersen, 1995). Such processors are named as DVFS enabled processors. By exploiting the DVFS feature, various energy efficient operating system scheduling algorithms were proposed for uniprocessor and multi-core platforms (Shin et al., 2001; Kim et al., 2002; Pillai and Shin, 2001; Shin and Kim, 2006; Chin, 2013; Seo et al., 2008; Devdas and Aydin, 2010; Lu and Guo, 2011; Khandhalu et al., 2011).

Another component of processor energy consumption is the static energy consumption which is present even when no logic operations are performed. The CMOS circuit technology is well known for its low static energy consumption. At the same time, there is a constant need of high performance and higher transistor density resulting in a continuous decrease in device dimensions in each technology generation (Borkar, 1999). As a result, there is constant electric field scaling which needs proportionate reduction in supply voltage. The reduction in supply voltage requires proportionate decrease in threshold voltage to maintain the desired gate delay. This leads to exponential increase in sub-threshold leakage current thereby giving rise to a significant amount of static energy consumption (Jejurikar et al., 2004). Dynamic Power Management (DPM) mechanisms like dynamic shutdown and procrastination can be used to reduce static energy consumption. DPM puts the processor in shut down mode whenever possible. The limitation of DPM is that it suffers from an overhead of mode switching which causes additional energy and latency penalty (Lee et al., 2003; Jejurikar et al., 2004; Niu and Quan, 2004; Chen and Kuo, 2007). Therefore, the processor is always turned to sleep mode whenever the idle interval is sufficiently larger than a certain threshold time duration called the breakeven time.

1.2.2 Hardware Platform

Due to increasing demand of higher processor performance and growing capacity for number of transistors after every 18 to 24 months as stated in Moore's law, processor designers focused on the circuit miniaturization to increase the clock frequency. But this has led to the problem of high energy consumption and excessive heat dissipation. For example, Intel canceled the launch of processor named Tejas in 2004 which was the successor of the Pentium P4 processor, due to its extremely high energy consumption. The problem of high energy consumption cannot be completely addressed by scaling the voltage and frequency alone as this would limit the maximum task execution frequency thereby restricting the performance. Therefore, in addition to DVFS, the solution to this problem also requires to use multiple cores on a single chip which can take better advantage of increasing transistor capacity and can achieve better performance by exploiting parallelism. In 2007, Intel released the first Core 2 Duo processor. Since then, there has been a paradigm shift towards the multi-core processors (Davis and Burns, 2011).

1.2.2.1 Classification of Multi-core Processors

With respect to the task scheduling, the multi-core processor platforms can be classified into three categories (Davis and Burns, 2011):

(1) *Homogeneous:* The processor cores are identical where the maximum operating frequency of all the processor cores is same.

(2) *Uniform:* The processor cores differ in their maximum operating frequency but it follows the same instruction set architecture (ISA). i.e., a task which executes at a speed of x on one core may execute at a speed of $2x$ on another core.

(3) *Heterogeneous:* The processor cores have different hardware configurations, frequencies, ISAs, private caches etc. The processor cores will not have inter-operability in executing tasks.

1.2.2.2 Memory Architecture

Memory architecture is an important aspect of any processor platform as it is one of the most energy consuming parts. Memory architecture is also vital for designing task scheduling algorithms on multi-core platforms which has private and shared spaces. The two main categories of memory architecture are distributed and shared memory architecture (Stallings, 2014). In distributed memory architecture, each core maintains its own local queue and as a consequence, a processor cannot directly access the data/instructions stored on another core. On the other hand, in shared memory architecture, as all the cores have access to a central shared memory space, any processor core can access the data/instructions belonging to the task of any other processor core.

The memory architecture has a direct impact on the latency of task migration from one core to another (Schirmeister, 2007). In distributed memory architecture, as each core maintains its own local memory, if a task is required to be migrated from one core to another, it has to transfer the entire task context including instructions and data to another core. This transition is costly in terms of time consumption. In shared memory architecture, the instructions and data of all the tasks are stored in a central memory and are therefore available to all the cores. But serving the read/write requests of all the tasks simultaneously is time consuming. Moreover, the size of central shared memory should be much larger than the local memories in distributed memory architecture as it has to store the instructions and data of all the tasks. This may lead to slower memory access. Thus a single central memory in shared memory architecture lowers the overall system performance and is not scalable with increase in number of processor cores.

In order to overcome the limitations of single central memory in shared memory architecture, modern processors make use of *hierarchical memory* architecture (Schirmeister, 2007). In this architecture, levels of small and fast local memories called *caches* are placed between processor and central memory. This type of memory organization helps to reduce the memory access latency. The instructions and data of the running task are stored in cache memory resulting in higher availability of data/instructions at any time instance provided the task has not migrated to another core. Since the cache memories are small and cannot store all the instructions and data corresponding to a running task, the requested data/instruction which is not present in cache memory is brought from the lower level memory.

In modern processors, there are multiple levels of cache memories. Level 1 cache (L1) is the smallest and fastest private cache which is nearest to the processor. Level 2 cache is bigger and slower than L1 cache. It may be private or shared amongst the cores. Both L1 and L2 are generally made up of SRAM. Level 3 cache (L3) is usually made up of SRAM and DRAM. It is larger and slower than L2 cache and it may be placed on or off the chip. Hence, as the read/write request goes down the memory hierarchy, it takes more time to transfer the data from/to the processor.

In hierarchical memory architecture with one or two levels of cache as local to the core, the transfer of task context in case of migration is time consuming. Additional time is required for reloading the data at the target core from the shared higher level cache or from the shared central memory and invalidating the data in present core. Thus, in both types of the memory architectures, migration of a task incurs significant overhead because of the memory access latency. In case of task preemption, the data / instructions of a new higher priority task gets loaded into all levels of cache memories. When the previous task resumes, its context may not exist in cache which results in increasing execution time of the task. Both preemption and migration of a task result in overhead which may affect the performance of the system. The proposed scheduling algorithms assume shared memory architecture with multiple levels of cache memories as shown in figure 1.

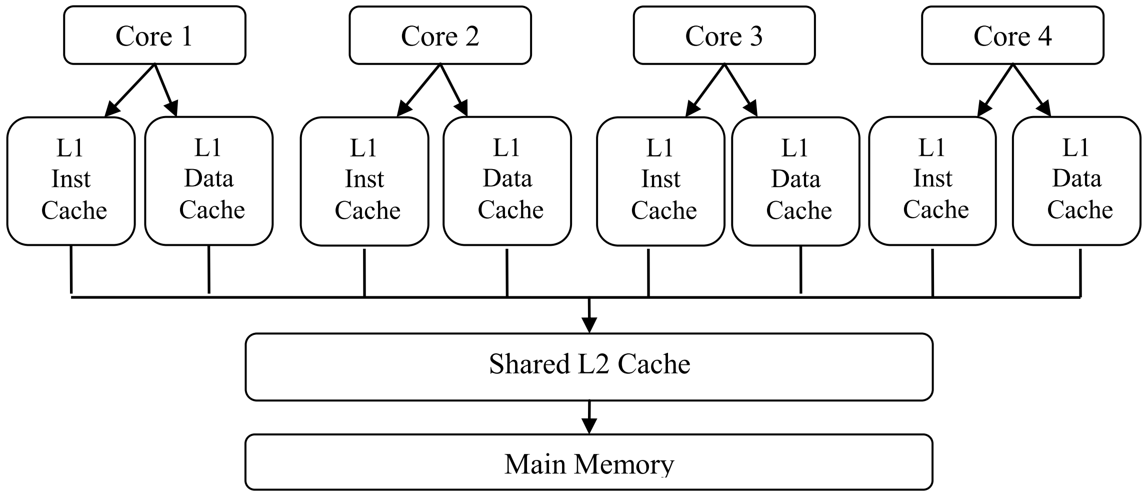


Figure 1.1: Hierarchical Memory Architecture

1.2.3 Real Time Task Model

In real time systems, the unit of work which is executed by the processor is known as *job* and the set of related jobs is called a *task*. The real time applications are composed of real time tasks. These tasks are executed under timing constraints. The real time constraints (or deadlines) are defined as either hard or soft based on the functional criticality of the tasks, usefulness of late results and deterministic and probabilities nature of the constraints. The distinction between hard and soft timing constraints is quantitatively stated as a function of *tardiness* of a job. The tardiness of a job measures the lateness of that job with respect to its deadline. The tardiness is zero if a job completes its execution on or before its deadline. If a job is late, its tardiness is the difference between the completion time and its deadline. A job with hard deadline falls abruptly and may even cause disaster if the tardiness of such jobs is greater than zero. On the other hand, the usefulness of the result produced by a soft deadline job decreases with increase in tardiness. Other attributes of a real time task are release time; period/inter-release time and worst case execution time (*wcet*). *Release time* is the time at which the job is available for execution and *period/inter-release time* is the time when next job of the task is released. *wcet* is the maximum amount of time required to complete the execution of the job. It mainly depends on the complexity of the job and speed of the processor. Since real time systems are deterministic in nature, the *wcet* of a task is known in prior through analysis and measurement but the actual execution time (*aet*) is not known. The *wcet* of a job is directly impacted by the structure of the program, type of processor, processor platform, memory architecture, communication network and the input data. The *worst case utilization* of a task is defined as the ratio of *wcet* and period of that task. It tells the percentage of processor time the task requires for its successful execution. The *total utilization* of a task set is the sum of utilizations of tasks in the task set. It is used for determining the schedulability of the task set. *Hyper period (H)* of a task set is defined as the least common multiple (LCM) of the periods of all the tasks in the task set. The number of jobs produced by the periodic tasks in each hyper period is equal to $\sum_{i=1}^{n} H/p_i$ where n is the number of tasks in a task set and p_i is the period of ith task (Liu, 2008).

There exist three types of real time tasks: (1) Periodic tasks (2) Aperiodic tasks and (3) Sporadic tasks. The *periodic tasks* are the tasks whose jobs arrive at regular intervals and they have hard deadlines. For example, in a radar system, a task of transmitting/receiving the radio signals for object detection is a periodic task. As discussed earlier, the real time system is required to respond to external events that arrive arbitrarily in the periodic task system; these external events are modeled using aperiodic and sporadic jobs whose release times are not known apriori. A task is *aperiodic* if its jobs have either soft deadlines or no deadlines. The task which adjusts the sensitivity setting of the radar surveillance system is an example of aperiodic task. Even though, these tasks have no deadlines, there late response is annoying. Hence, it is important to optimize the responsiveness of the aperiodic jobs but never at the expense of hard deadline periodic jobs. In contrast, the tasks that arrive arbitrarily but have hard deadlines are called *sporadic tasks* (Liu, 2008). For example, in an automatically controlled train, if the task of applying brakes for stopping the train is not done as soon as the command to do so is made, it may cause disaster. In this case, the command to apply the brake is a sporadic job which arrives arbitrarily but has hard deadline.

In this thesis, the proposed energy efficient scheduling algorithms schedule mixed task sets that contain periodic and aperiodic tasks. These scheduling algorithms optimize energy consumption along with achieving good response time of aperiodic tasks without missing the deadlines of periodic tasks.

1.2.4 Energy Efficient Real Time Task Scheduling

The energy efficient task scheduling can be implemented for processors that are equipped with discrete frequency and voltage settings and are capable of modifying the frequency and voltage dynamically at run time. The majority of the modern embedded systems use such processors (Chen and Kuo, 2007).

DVFS is an effective way of reducing the dynamic energy consumption of low power embedded systems (Kim et al., 2002). It adjusts the supply voltage and corresponding clock frequency of the processor dynamically without affecting the performance of the system. Since the energy consumption of CMOS based processor has quadratic dependence on supply voltage, lowering the supply voltage effectively reduces the energy consumption. Therefore, DVFS is popularly used to reduce the processor level energy consumption of the real time embedded systems (Chen and Kuo, 2007). It is integrated with the real time task scheduling where at each scheduling decision point, the frequency and voltage of the processor is decided. Such task scheduling algorithms are called DVFS based energy efficient task scheduling algorithms.

There exists another technique known as dynamic power management (DPM) for reducing static energy consumption (Jejurikar et al., 2004). Static energy dissipation takes place for all the time except when processor is in shutdown state. According to DPM technique, the processor can be dynamically shutdown when it is in idle state. However, putting the processor in shutdown state and then waking it up incurs some overhead because the processor loses temporal data stored in various forms of memory such as registers, caches, TLBs etc. Thus, before shutting down, all registers must be saved and dirty cache lines must be written back to the memory and upon wake up, all the saved data must be retrieved back to registers, cache lines etc. This results in additional memory accesses and hence additional energy consumption. In order to decide whether to shut the processor down or not, idle threshold interval is computed based on the idle state energy consumption and shutdown overhead. If the idle interval is less than the threshold, then it is not energy efficient to shutdown the processor.

1.3 Research Objectives

The aim of this research is to design and develop an energy efficient real time task scheduling framework for mixed task model containing a mix of hard and soft real time tasks on homogeneous multi-core processor platform. In order to achieve the aim of this research work, the following objectives were planned:

1. Design and implementation of energy efficient real time scheduling algorithms for mixed task sets on uniprocessor platform.
2. Design and implementation of energy efficient real time scheduling algorithm for dynamic energy optimization considering multi-core processor platform and mixed task sets.
3. Design and implementation of a real time scheduling algorithm for the overall energy optimization which includes both static and dynamic energy consumed by multi-core processor for mixed task sets.

1.4 Methodology

The objectives defined in the previous section are achieved by the accomplishment of the following:

1. Study and analysis of the existing energy efficient real time scheduling algorithms for uniprocessor as well as multi-core/multi-processor platforms.
2. Generation of synthetic task sets containing both periodic and aperiodic tasks for various utilizations, number of tasks and hyper periods.
3. Design, implementation, validation and analysis of energy efficient scheduling algorithms for uniprocessor platform.
4. Design, implementation, validation and analysis of scheduling algorithm for multi-core platform.
5. Design, implementation, validation and analysis of scheduling algorithm for dynamic energy optimization for multi-core platform.
6. Design, implementation, validation and analysis of a scheduling algorithm for the overall energy optimization which includes both dynamic and static energy for multi-core platform.
7. Design and development of a full-fledged software tool which includes the implementation of basic real time scheduling policies, existing energy efficient

scheduling policies, the scheduling algorithms proposed during the thesis work and the task set generation algorithm.

1.5 Thesis Organization

The structure of the thesis is illustrated in figure 1.2. This thesis is composed of six chapters.

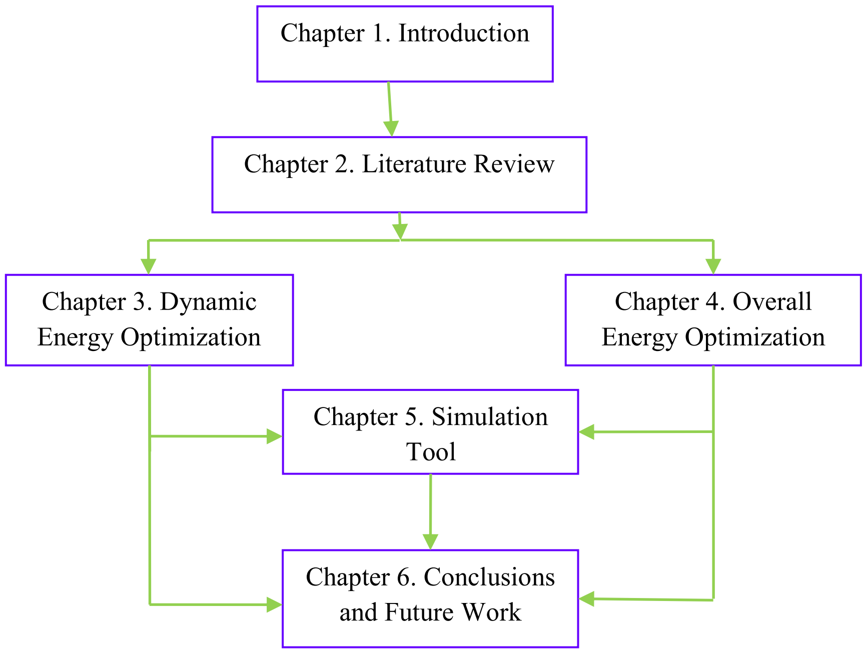

Chapter 1. Introduction
Chapter 2. Literature Review
Chapter 3. Dynamic Energy Optimization
Chapter 4. Overall Energy Optimization
Chapter 5. Simulation Tool
Chapter 6. Conclusions and Future Work

Figure 1.2: Thesis Organization Roadmap

Chapter 2 provides an overview of on-going research directions in energy aware task scheduling for uniprocessor, multi-core and multi-processor platforms. It reflects the diversity of the issue involved in making the task scheduling energy efficient. It discusses the fundamental study of energy optimization techniques, structure and function of multi-core processors etc that are necessary in this research work. Together, it provides a comprehensive overview of the existing literature and open issues to provide foundation for this research work.

Chapter 3 presents the scheduling algorithms developed for reducing dynamic energy. It describes the DVFS based scheduling algorithms for mixed task set for uniprocessor and multi-core platforms. The algorithms discussed in this chapter are able to reduce dynamic energy consumption of the processor by meeting all the hard deadlines of the periodic tasks and ensuring early response times of aperiodic tasks. But these algorithms do not take care of reducing the static energy consumption that exists in processor's idle state.

Chapter 4 further extends the algorithms described in chapter 3 by reducing the static energy consumption along with the dynamic energy consumption. It integrates DPM and procrastination techniques with DVFS technique to reduce the overall energy consumption. The results and analysis show that there is further reduction in processor energy consumption. The hard deadline of periodic tasks and responsiveness of aperiodic tasks are taken care in addition to reduction in overall energy consumption.

Chapter 5 illustrates the details of design and implementation of the software simulation tool called "STREAM". STREAM stands for "Simulation Tool for Real time Energy efficient scheduling and Analysis for Multi-core processors". It provides the in-depth explanation of the architecture of the simulation tool, implementation of various scheduling algorithms and various other novel aspects that are implemented in STRAEM which are missing in other existing scheduler simulators.

Chapter 6 concludes the thesis by discussing the overall contribution of the research in the context of the related work in this area. In addition, it discusses limitations of the work and points to future research directions.

Chapter 2

Literature Review

This chapter provides an overview of previous research work on energy efficient real time scheduling algorithms. It introduces a variety of energy efficient scheduling algorithms existing in the literature on uniprocessor, multi-core and multiprocessor platforms. The real time task models considered in this literature review are periodic task model and mixed task model where periodic and aperiodic tasks are scheduled together. The main focus of the research described in this thesis is the energy efficient real time scheduling algorithms for mixed task model on multi-core platform.

2.1 Introduction

In the last decade, we have witnessed a paradigm shift in the embedded systems domain. The growing demand of portable and battery operated high performance embedded devices has essentially motivated the researchers to concentrate on two important aspects: energy optimization and multi-core processors.

The problem of energy consumption has been addressed at various levels - architecture level, operating system level, compiler level, application program level and system program level (Saha and Ravindran, 2012). At the architecture level, energy consumption can be reduced by improving instruction set architecture, optimizing the memory subsystem and by managing I/O operations more efficiently. At the operating system level, energy consumption can be reduced by improving various aspects of operating systems such as task scheduling, inter-process communication, paging systems etc. Similarly, the energy consumed by application and system programs can be reduced by efficient use of data structures and by optimizing the programs using compilers.

Another major change in hardware design that can help in reducing energy consumption is the parallel execution of tasks on multiple cores rather than executing them sequentially on one processor at a very high speed. This is possible due to increasing growth of number of transistors on a fixed die size. The hardware designers realized that increasing the complexity of the hardware increases the performance at the cost of high energy consumption. Therefore, the trend has changed towards utilizing large number of transistors for constructing multiple cores/processors on a fixed die size thereby exploiting parallelism and reducing energy consumption.

In this chapter, we focus on energy saving mechanisms handled at the operating system level (particularly by using real time task scheduling) on uniprocessor, multi-core and multiprocessor platforms.

2.2 Background

We present the research carried out in the area of energy aware real time scheduling in last two decades. The literature can be classified on the basis of the different processor platforms and the type of energy that is targeted for optimization. Early research in late 1990's focused on energy optimization for uniprocessor platform. Later, after year 2000, majority of the research in energy aware real time scheduling focused on multi-processor platform. In recent time (after year 2008 till now), researchers are working on different issues revolving around energy optimization on multi-core platform.

2.2.1 Classification of Energy Aware Scheduling

Irrespective of the processor platforms, the existing research can be classified into two broad categories based on whether the scheduling techniques focus on reduction of dynamic energy consumption alone or on both dynamic and static energy consumption. Figure 2.1 shows taxonomy of the literature discussed in this chapter.

2.2.2 Dynamic Energy Optimization

The tradeoff between performance and energy consumption can be addressed using energy optimization technique called DVFS. DVFS reduces dynamic energy consumption without hampering the performance of real time application. It is based on two important aspects:

- The modern CMOS based processors are equipped with multiple discrete voltage and frequency levels and are capable of varying operating frequency and supply voltage dynamically.

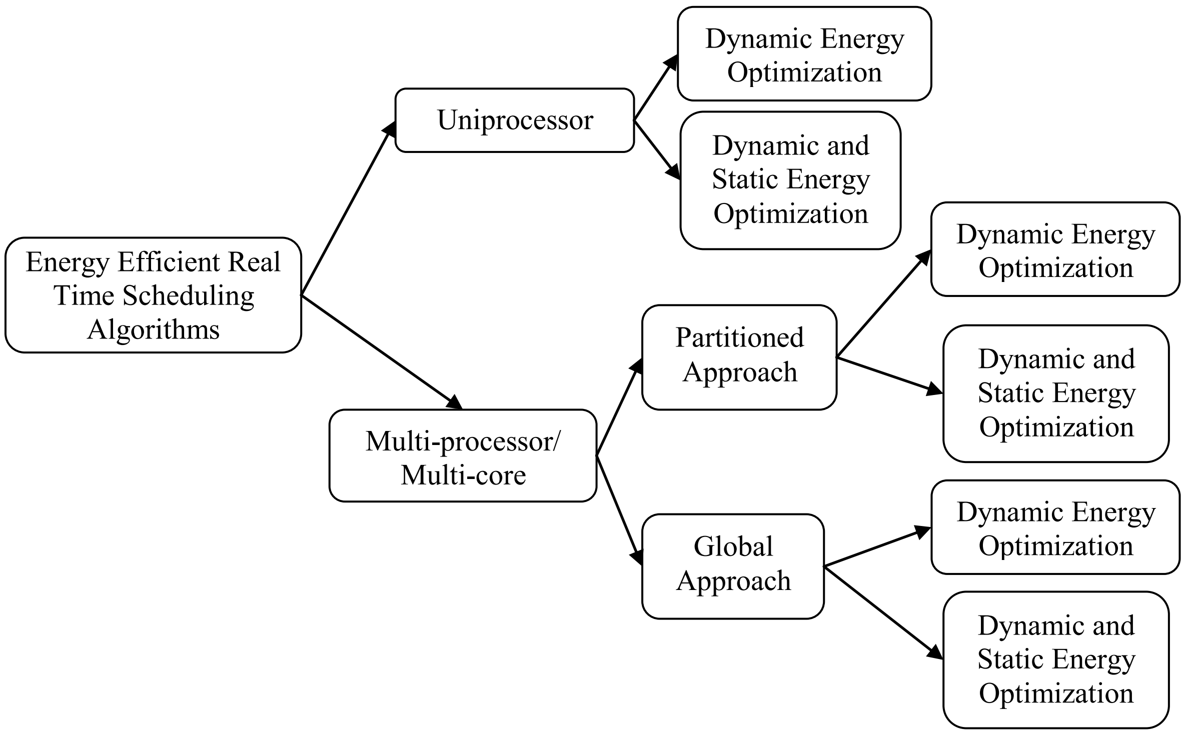

Energy Efficient Real Time Scheduling Algorithms
Uniprocessor
Dynamic Energy Optimization
Dynamic and Static Energy Optimization
Multi-processor/ Multi-core
Partitioned Approach
Dynamic Energy Optimization
Dynamic and Static Energy Optimization
Global Approach
Dynamic Energy Optimization
Dynamic and Static Energy Optimization

Figure 2.1: Taxonomy of Energy Aware Real Time Scheduling

- In majority of applications, high performance (or high frequency/speed) is required only for small fraction of time while rest of the time we can have low performance (or low frequency/speed).

DVFS technique works on the idea of varying operating frequency of processor which in turn varies supply voltage (Weiser et al., 1994). It lowers the frequency when the processor load (or utilization) is low and increases the frequency when the processor load (or utilization) increases. It exploits the fact that *aet* is most of the time less than or equal to *wcet* of a task. The difference between *wcet* and *aet* is called slack time. This available slack time is used for slowing down the processor to save energy. Since energy consumed per cycle with CMOS circuitry scales quadratically with the supply voltage, DVFS potentially reduces the processor energy consumption through voltage and frequency scaling. In order to decide, when to execute at higher frequency and when at lower frequency, it requires the cooperation of operating system scheduler with voltage and frequency selection circuit. Therefore, DVFS technique is always applied in coordination with operating system scheduler.

When DVFS techniques are applied to submicron or deep submicron regimes, it leads to increase in static energy consumption, resulting in increase of total energy consumption (Lee et al., 2003; Jejurikar and Gupta, 2004). This is because, slowing down the processor execution beyond certain frequency increases leakage current which leads to increase in static energy consumption and thus increases total energy consumption. The operating frequency below which static energy consumption dominates dynamic energy consumption resulting in increase in total energy consumption is called *critical speed* (Jejurikar et al., 2004). Thus, DVFS technique is more energy efficient if it does not scale down below critical speed. If we assume that normalized operating frequency lies within a range of 0 to 1, then the critical slow down factor (η_{crit}) can be defined as ratio of critical speed (or frequency) to maximum operating frequency. If slow down factor (η_i) computed by any DVFS technique is less than η_{crit} then the computed slow down factor is raised to η_{crit} to save static energy consumption.

In real time embedded systems, the real time tasks should be executed under timing constraints. These tasks have hard or soft deadlines before which they should finish execution. Application of DVFS techniques on real time scheduling algorithm is challenging because slowing down the processor speed should not cause a task to miss the deadline.

2.2.3 Overall Energy Optimization

DVFS limits energy optimization below critical speed due to the dominance of static energy consumption. In order to overcome this limit and reduce static energy consumption, shutting down the processor when it has enough idle time is proposed (also known as DPM mechanism) (Lee et al., 2003; Jejurikar and Gupta, 2004). As shutting down the processor during idle period may incur overhead, it is required to set a threshold called break even time to decide whether it is energy efficient to shutdown or not. For example, the breakeven time of a 70nm Transmeta Crusoe processor is 2 msec (Jejurikar and Gupta, 2004). In order to stretch the idle period to reduce the number of short shutdown intervals and also to reduce the shutdown overhead, a technique called procrastination can be used (Lee et al., 2003; Jejurikar et al., 2004) which delays the execution of jobs that arrive during idle period to increase the span of idle interval. In this way, overall energy optimization can be achieved by the combination of slowdown, shutdown and procrastination techniques.

Identification of idle and procrastination intervals require the cooperation of operating system scheduling algorithm. Thus, both DVFS and DPM with procrastination techniques require the cooperation of scheduling algorithm. Both types of techniques are applied on real time scheduling algorithms to minimize the overall energy consumption of real time embedded systems.

2.3 Scheduling Algorithms for Uniprocessor Platform

We start the review of research on how developments in energy efficient scheduling algorithms progressed during late 1990s on uniprocessor platform. This was an era when low power CMOS based microprocessors were introduced (Chandrakasan et al., 1992; Younis and Knight, 1993; Weiser et al., 1994; Burd and Brodersen, 1995). The idea of energy optimization by variable voltage and frequency was initially introduced by Weiser et al. (1994). According to Weiser, DVFS based scheduling techniques utilize the dynamic slack time generated by the running job upon its completion. Particularly for real time systems, DVFS takes advantage of the fact that once the real time requirements of the real time tasks are met; there is no further advantage in increasing the throughput. Therefore, voltage and frequency scaling techniques take care of meeting the timing constraints of the real time tasks.

2.3.1 Dynamic Energy Saving

This section presents the literature on dynamic energy saving using energy aware real time scheduling algorithms specifically designed for uniprocessor platform. The algorithms designed for periodic and mixed task models are discussed separately.

2.3.1.1 Periodic Task Model

There exist various DVFS based real time scheduling algorithms in the literature which show different ways to utilize and distribute the slack time. Kim et al. (2002) have done a comparative study of existing energy efficient scheduling algorithms for periodic task model where they classified the existing algorithms based on the way slack time is estimated and distributed. They classified the algorithms into three categories: intra-DVFS, inter-DVFS and hybrid approach.

Intra-DVFS (Shin et al., 2001; Gruain et al., 2001) algorithms adjust the frequency and corresponding voltage by utilizing the available slack time within the task boundary. Inter-DVFS (Kim et al., 2002) algorithms determine the frequency/voltage on task by task basis at each scheduling point. They distribute the slack time from the current task for the tasks following it. The hybrid approach follows a mix of these two strategies.

Intra-DVFS algorithms are based on the execution paths taken by the application during run time. When the execution path taken is not the worst case execution path, it results in slack time. This slack time is used to adjust the frequency and voltage of the processor. There are two methods under Intra-DVS algorithms: *Path-based method* and *Stochastic method*. In Shin et al. (2001), the authors described the path-based Intra-DVS method in which initially the execution path is set to worst case execution path. When the actual execution path deviates from the initially fixed path, for example, by branch instruction, the slack time is generated and this slack time is utilized for voltage and frequency scaling. The program locations are identified using static program analysis and execution time profiling (Shin et al., 2001; Lee and Sakurai, 2000). In Gruain et al. (2001), the authors described the stochastic method in which the application is initially started at a low speed and the execution is later accelerated if needed. If the task takes less than the *wcet*, then the speed is not raised. The speed is raised or slowed regardless of the execution paths taken at run time. As compared to the path-based method, this method does not utilize the slack time effectively. Kim et al. (2002) state that the path-based Intra-DVFS achieves better performance than stochastic method when the slack time is limited where as in case of large amount of slack time, the stochastic method performs better.

Kim et al. (2002) define the Inter-DVFS algorithm as "*run-calculate-assign-run*" strategy to determine the frequency and voltage which can be elaborated as follows: (1) *run* the current task (2) upon completion of current task, *calculate* the maximum allowable execution time of the next scheduled task (3) *assign* the frequency and voltage to the next task (4) *run* the next task. The Inter-DVFS algorithms differ in performing step 2 that computes the maximum allowed execution time which includes the *wcet* and available slack time.

The inter-DVFS algorithms consist of two parts (Kim et al., 2002): *slack estimation* and *slack distribution*. *Slack estimation* identifies the maximum available slack time from the tasks whereas *slack distribution* distributes the identified slack time uniformly among the tasks so that the speed is uniformly reduced during the schedule. Many inter-DVFS algorithms use greedy approach for slack distribution where all the available slack time is given to the next ready task. It is widely used because of its simplicity. In slack estimation based inter-DVFS algorithms, there exist two sources of slack times: *static slack time* and *dynamic slack time*. Static slack time is the extra time available for the next task that can be statically identified. Dynamic slack time is identified at run time during task execution.

The slack estimation and slack distribution methods are different for periodic and mixed task systems. For periodic task systems, maximum constant speed is a static slack estimation method in which lowest possible clock speed that feasibly schedules the task set is calculated based on the worst case utilization of the task set. If the total task set utilization U is less than 1.0 at maximum frequency f_{max}, then the frequency of execution is lowered to U*f_{max} when the task is scheduled using Earliest Deadline First (EDF) (Kim et al., 2002; Pillai and Shin, 2001). This method cannot identify dynamic slack time that exists during the task execution. Kim et al. (2002) proposed three dynamic slack estimation methods for scheduling periodic task set: Stretching to Next Task Arrival (SNTA), Priority Based Slack Stealing (PBSS) and Utilization Updating (UU).

SNTA estimates slack time of the ready task using its next task arrival time (NTA). Assume the ready task T is scheduled at time t. If the task's NTA is later than (t + *wcet*(T)), then the task T can be executed at a lower frequency such that it completes exactly at its NTA provided the ready queue does not contain any other task. The algorithms low power priority based scheduling Earliest Deadline First (lppsEDF) and low power priority based scheduling Rate Monotonic (lppsRM) (Shin et al., 2000) are based on this approach.

In PBSS, the slack time generated by early completion of the higher priority task can be used by the following lower priority task to lower the execution frequency. The algorithms Dynamic Reclamation Algorithm (DRA) and Aggressive (AGR) are based on PBSS (Aydin et al., 2001, Aydin et al., 2004). In UU, the total processor utilization is recalculated using *aet* of the completed task at each scheduling point so that frequency/voltage can be scaled accordingly. The algorithms cycle conserving Earliest Deadline First (ccEDF) and look ahead Earliest Deadline First (laEDF) are based on this scheme (Pillai and Shin, 2001). The main advantage of this method is its simple implementation since it has to only update the current processor utilization at each scheduling point. Saha and Ravindran (2012) have done experimental evaluation of many of the existing algorithms mentioned above such as ccEDF, laEDF, DRA, AGR etc. The experimental evaluation is done using the ChroneOS, a real time Linux kernel (Dellinger et al., 2011) on two hardware platforms: ASUS intel-i5 processor and AMD Zacate mini-ITX motherboard.

2.3.1.2 Mixed Task Model

The inter-DVFS algorithms for periodic task systems can be directly applied to mixed task systems only if aperiodic tasks execute at maximum frequency as slack time of aperiodic task is not known. Otherwise, there is a need of different slack estimation schemes where aperiodic tasks are allowed to execute at scaled frequency. Irrespective of whether aperiodic task is executed at maximum or at scaled frequency, the arbitrary temporal behavior of the aperiodic tasks requires one to judiciously select the aperiodic server in order to achieve better response time and reduced energy consumption.

Various bandwidth preserving servers like Deferrable Server (DS), Sporadic Server (SS), Total Bandwidth Server (TBS) and Constant Bandwidth Server (CBS) have been proposed by researchers and are popularly used to schedule mixed task sets (Lui, 2008). Since these servers limit aperiodic task to execute within the available bandwidth, these algorithms can estimate slack times by using characteristics of bandwidth preserving servers. Along with slack estimation, these servers have to ensure energy efficiency as well as responsiveness of aperiodic tasks. Shin and Kim (2006) proposed four slack estimation schemes, namely, Stretching to Next Replenishment Time (SNRT), Bandwidth Based Slack Stealing (BBSS), Periodic Only Slack Distribution (POSD) and Workload based Slack Estimation (WSE).

SNRT is a modified SNTA scheme which stretches the execution time of the ready task till arrival time of next task and is applied on DS (Shin and Kim, 2004) and SS (Shin and Kim, 2006). While executing an aperiodic task, if there is no periodic task in the ready queue then currently executing aperiodic task can be stretched to min(NTA,R) where R is replenishment time of aperiodic task. If there is only one periodic task in ready queue and the server budget (qs) of aperiodic task is 0, then the periodic task can be stretched to min(NTA,R). Shin and Kim (2006) proposed SS/lpps-RMS which is based on lpps-RM algorithm (Shin et al., 2000) that uses SNRT scheme with SS for mixed task set. SNRT gives poor energy performance if aperiodic workload is small. This is because of the constraint that the periodic task cannot be stretched if qs > 0 which is true most of the time when aperiodic workload is very low. In order to overcome this limitation, BBSS scheme is proposed (Shin and Kim, 2006).

BBSS scheme handles this case by computing slack time as the amount of time available between current time and NTA of the periodic task excluding remaining execution budget in qs. Shin and Kim (2006) proposed SS/lppsRM-B algorithm which is based on BBSS. SS/lppsRM-B is based on lppsRM that uses BBSS with SS for mixed task set.

Periodic Only Slack Distribution (POSD) scheme is simpler than SNRT and BBSS schemes but offers better responsiveness (Shin and Kim, 2006). In SNRT and BBSS schemes, the periodic as well as aperiodic tasks use slack time to scale down the frequency but in POSD scheme, the entire slack time is utilized by periodic tasks only and aperiodic tasks are always executed at full speed. In POSD, the slack time is estimated using BBSS. This scheme gives better response time with slight degradation in energy saving. The lppsEDF-P algorithm is based on POSD scheme (Shin and Kim, 2006). Workload based Slack Estimation (WSE) scheme is applied on CBS. Since CBS does not have fixed intervals, stretching rules of SNRT, BBSS and POSD cannot be applied on aperiodic task. Here slack time is identified when workload of CBS is less than the server utilization. CBS/DRA-W algorithm is based on WSE scheme (Shin and Kim, 2006).

Aydin and Yang (2004) proposed a composite performance metric, Energy * Average Response Time. They also proposed three slack reclamation schemes which are applied on DRA with TBS. These schemes are basic reclamation scheme (BRS), mutual reclamation scheme (MRS) and bandwidth sharing scheme (BSS). BRS and MRS dynamically use the slack time while BSS aggressively utilizes the bandwidth of TBS for slowing down periodic tasks. Kuo (2013) proposed an algorithm, ratio based aggressive reclaim algorithm (RARA), which is based on DRA and TBS. The slack reclamation method of RARA performs better than MRS and BSS for the composite metric of energy and average response time. Min-Allah et al. (2008) proposed an algorithm in which deadline of TBS is extended based on available server utilization and periodic tasks are executed at static frequency. Digalwar et al. (2013) proposed an energy efficient DVFS algorithm EEDVFS that uses UU method for scheduling periodic tasks. The aperiodic tasks are executed at maximum frequency to achieve reduced response time. The real time scheduling policies used are EDF and DS.

There are few other algorithms like Greedy Reclamation of Unused Bandwidth - Power Aware (GRUB–PA) (Scordino and Lipari, 2006), On-Line DVS algorithm (OLDVS) (Lee and Shin, 2004) which follow different methods other than the methods mention above.

Wu and Wu (2014) proposed an energy efficient algorithm for scheduling dependent real time tasks which are required to access multi-unit resources in a system. Niu and Quan (2015) considered energy consumed by peripheral devices along with processor energy consumption and proposed a scheduling algorithm which minimizes system wide energy consumption. They considered weakly hard real time systems where at least m jobs should meet deadline out of k jobs.

Table 2.1 shows the summary of the uniprocessor based energy efficient scheduling algorithms discussed above under various voltage and frequency scaling techniques for periodic and mixed task model.

The limitation of the energy efficient scheduling algorithms that reduce dynamic energy consumption is that they substantially lead to increase in static energy consumption which is caused due to leakage current. Many researchers have focused on leakage aware DVFS scheduling algorithms on periodic task systems for different processor platforms.

Table 2.1: Classification of DVFS techniques and the target uniprocessor scheduling algorithms

Task Model	Broad Categories of DVFS Techniques	Voltage and Frequency Scaling Methods	DVFS based Scheduling Algorithms
Periodic Task Model	Intra-DVFS	Path based method	intraShin (Shin et al., 2001)
		Stochastic method	intraGruian (Gruian et al., 2001)
	Inter-DVFS	Maximum Constant Speed	Static Voltage Scaling based EDF and RM (Pillai and Shin, 2001)
		Stretching to NTA	lppsEDF and lppsRM (Shin et al., 2000)
		Priority based Slack Stealing	DRA and ARG (Aydin et al., 2001, Aydin et al., 2004)
		Utilization Update	ccRM, ccEDF and laEDF (Pillai and Shin, 2001)
Mixed Task Model (Periodic and Aperiodic)	Inter-DVFS	Stretching to NRT	SS-lppsRM (Shin and Kim, 2006), DS-lppsRM (Shin and Kim, 2004), SS-ccRM, DS-ccRM
		Bandwidth Based Slack Stealing	SS-lppsRM-B, DS-lppsRM-B, SS-ccRM-B and DS-ccRM-B (Shin and Kim, 2006)
		Periodic Only Slack Distribution	CBS-lppsEDF (Shin and Kim, 2006), CBS-DRA (Shin and Kim, 2006), TBS-BRS, TBS-MRS and TBS-BSS (Aydin and Yang, 2004), RARA-TBS (Kuo, 2013)
		Workload based Slack Estimation	CBS-DRA-W (Shin and Kim 2006)
		Hybrid Method	EEDVFS (Digalwar et al., 2013)

2.3.2 Overall Energy Saving

As the speed of processor is reduced below the threshold speed, static energy consumption becomes dominant resulting in increase in overall energy consumption. In DVFS enabled processors, the static energy is consumed in two situations: one, when the processor is in idle state and the other, when execution speed is reduced below critical speed. Therefore, there is a need to reduce static energy along with dynamic energy consumption. This section provides the details of the scheduling policies that minimize both dynamic and static energy consumption for uniprocessor platform.

Martin et al. (2002) proposed a scheduling algorithm that uses DVFS along with adaptive body biasing to reduce overall energy consumption and derived an analytical expression to compute power consumption and processor performance as a function of frequency, supply voltage and body bias voltage. In Lee et al. (2003), the authors proposed a software controlled leakage power technique LC-EDF was investigated in which the tasks that arrived during idle interval were procrastinated and processor was put in shutdown mode. For rest of the time, the processor executes at maximum frequency. Procrastination determination was done on the basis of *wcet* of the tasks. In (Jejurikar and Gupta, 2004; Jejurikar et al., 2004), the authors identified an operating point, called critical speed, below which it is not energy efficient to run the processor. Their algorithm minimized overall energy consumption by applying DVFS and static procrastination. They assumed that the processor is accompanied by a controller which handles task procrastination. Niu and Quan (2004) proposed an algorithm, DVSLK, which is similar to the algorithms in (Jejurikar et al., 2004) but with a difference in the method of calculation of procrastination interval. The algorithm DVSLK uses job based strategy to compute job's delay interval. This is more accurate than the task based strategy used in (Jejurikar et al., 2004). Jejurikar and Gupta (2005) extended their previous work by dynamically reclaiming the slack time generated by completed jobs. Their algorithm maintains a free run time list which contains the details of jobs with their slack time in priority order. This slack time is included in the procrastination interval to lengthen the delay interval.

Chen and Kuo (2006) proposed an energy efficient rate monotonic scheduling algorithm which is designed in two phases. In the first phase, execution frequency and supply voltages of the tasks are determined by applying an offline method and in the second phase, the event at which processor is turned on/off is determined. Chen and Kuo (2007a) proposed an algorithm in which instead of applying procrastination greedily, a parametric procrastination is done. This technique delays the job execution only if the interval is large enough for the processor to be shutdown and provides substantial energy saving. Chen and Thiele (2008) adopted accelerating frequency strategy from (Lorch and Smith, 2004) to reduce dynamic energy and modified it to save static energy by scaling the frequencies of jobs to critical frequency if the selected frequency is less than critical frequency.

Huang et al. (2009) considered systems with active, standby and sleep mode each with different energy consumptions. They developed algorithms for activation and deactivation of the processor using historical information of event streams. The tasks are considered as event streams whose arrivals are calculated using real time calculus with hard timing constraints. A controller is used to make decisions during sleep mode. Nui (2010) worked on ensuring QoS along with minimizing the overall energy consumption on soft real time tasks. The QoS guarantee is achieved using window constraints which require at least m jobs out of k non-overlapped jobs of a task to meet the deadline. The scheduling technique is based on pattern variation, dynamic slack reclamation and procrastination. Awan and Petters (2011) proposed an algorithm which reduces only static energy consumption by accumulating the static and dynamic slack time by executing tasks at maximum frequency. Chen et al. (2013) considered arbitrary event arrivals and modeled these arrivals using real time calculus as in (Huang et al., 2009). They proposed an online algorithm, namely Optimal Workload Aware Algorithm (OWAA) which determines the optimal time for waking up the processor from sleep mode and optimal frequency to execute the job after wake up. Arrival curve model is used to estimate worst case idle interval.

Table 2.2 provides the summary of leakage aware uniprocessor scheduling algorithms. It can be observed from the literature discussed above and from table 2.1 and 2.2 that energy efficiency is sufficiently addressed in uniprocessor platform for periodic task model. For the mixed task model, dynamic energy optimization is well researched but the overall energy reduction is not thoroughly addressed.

Table 2.2: Summary of Leakage Aware Uniprocessor Scheduling Algorithms

Broad Classification of Parameters	Parameters of Comparison Under Broad Category	References
Type of Energy	Static Energy	Lee et al., (2003), Haung et al., (2009), Awan and Petter (2011)
	Static and Dynamic Energy	Martin et al., (2002), Jejurikar and Gupta (2004), Jejurikar et al., (2004), Niu and Quan (2004), Jejurikar and Gupta (2005), Chen and Kuo (2006), Chen and Kuo (2007), Chen and Thiele (2008), Niu (2010), Chen et al., (2013)
Task Model	Periodic Task Model	Martin et al., (2002), Lee et al., (2003), Jejurikar and Gupta (2004), Jejurikar et al., (2004), Niu and Quan (2004), Jejurikar and Gupta (2005), Chen and Kuo (2006), Chen and Kuo (2007), Chen and Thiele (2008), Niu (2010)
	Sporadic Task Model	Haung et al., (2009), Awan and Petter (2011), Chen et al., (2013)
Method of Validation	Simulation Study	Lee et al., (2003), Jejurikar and Gupta (2004), Jejurikar et al., (2004), Niu and Quan (2004), Jejurikar and Gupta (2005), Chen and Kuo (2006), Chen and Kuo (2007), Chen and Thiele (2008), Haung et al., (2009), Niu (2010), Awan and Petter (2011), Chen et al., (2013)
	Experimental Study	Martin et al., (2002)

2.4 Scheduling Algorithms for Multiprocessor Platform

In this section, we first present basic approaches of real time scheduling algorithms for multi-processor systems and then discuss about DVFS and leakage aware scheduling algorithms existing in the literature.

2.4.1 Non Energy Aware Scheduling

Multi-processor real time scheduling algorithms are categorized into partitioned and global approaches. Both the approaches are applied on fixed priority as well as dynamic priority periodic task systems. A survey on EDF and RM based partition and global multiprocessor scheduling algorithms for periodic task systems is done in (Gracioli et al., 2013; Davis and Burns, 2011). These papers discuss partition based algorithms such as Rate Monotonic First Fit (RMFF), Rate Monotonic Worst Fit (RMWF), Earliest Deadline First - First Fit (EDF-FF), Earliest Deadline First - Best Fit (EDF-BF) and Earliest Deadline First - Worst Fit (EDF-WF) and global algorithms based on RM and EDF.

2.4.1.1 Periodic Task Model

Davis and Burns (2011) gave detailed analysis of various existing scheduling algorithms for periodic task system. They discussed existing partitioned and global dynamic priority scheduling algorithms such as RMFF, RMWF, EDF-FF, EDF-BF, Proportionate fair (Pfair), Early Release Fair (ERFair), LLREF etc. As stated in the literature, partitioned scheduling have utilization bound of 50% (Zapata and Alvarez, 2004; Lopez et al., 2004; Baruah, 2013, Cho et al., 2006) and global scheduling, even if it gives utilization bound of 100%, generates a large number of preemptions, incurring significant overhead (Davis and Burns, 2011; Baruah et al., 1996). To overcome the difficulties of partitioned and global approaches, researchers have proposed another category of algorithms called semi-partitioned scheduling algorithms (Anderson and Tovar, 2006; Lakshmanan et al., 2009; Guan et al., 2010; Sousa et al., 2011). In this approach, one or few tasks from a task set are split into sub tasks. These sub tasks are assigned globally to two or more processors and rest of the tasks are partitioned. With this approach, utilization bound of partitioned approach is improved and preemptions are also reduced due to restricted global assignment.

2.4.1.2 Mixed Task Model

Very few scheduling algorithms have been proposed for mixed task systems. Baruah and Lipari (2004a) proposed an algorithm that uses excess capacity of processors for executing aperiodic tasks whereas periodic tasks are scheduled using global approach. They proposed another algorithm which uses constant bandwidth server to schedule aperiodic tasks (Baruah and Lipari, 2004b). Kato and Yamasaki (2008) proposed an algorithm in which aperiodic jobs are assigned to a processor where they are executed completely. This helps in reducing their response time. On the other hand, periodic jobs are allowed to migrate upon preemption. The global and partitioned scheduling algorithms are proposed in (Anderson et al., 2003a; Anderson et al., 2003b) for aperiodic tasks whose future arrivals are unknown. Brandenburg and Anderson (2007) proposed an algorithm that handles a mix of hard real time tasks, soft real time tasks and best effort aperiodic jobs. Hard real time tasks are partitioned and scheduled using EDF. Soft real time tasks and best effort tasks are scheduled in the background using dynamic slack reclamation techniques. Another scheduling algorithm for scheduling aperiodic tasks uses global approach in which priority assignment is done on the basis of slack time to avoid dhall effect (Lundberg and Lennerstad, 2008). The algorithm proposed in (Tang et al., 2011) can schedule periodic tasks along with aperiodic tasks on heterogeneous multi-resource systems. Digalwar et al. (2014) proposed an algorithm for multi-core processor platform for mixed task sets where periodic tasks are allocated to different cores using Bin Packing algorithm, WFD. The aperiodic tasks follow global approach in which an arriving aperiodic task is assigned to a core which can result in earlier response time. In this algorithm, aperiodic tasks utilize the excess capacity of the processors that is remaining after the periodic task allocation. This is an example of hybrid approach in which excess capacity of the cores which exist after partitioning is utilized by applying global approach.

2.4.2 Dynamic Energy Saving

We now discuss DVFS based multi-processor scheduling algorithms for periodic as well as mixed task models. One of the early works on energy saving on multiprocessor system was carried out by Zhu et al. (2003) in which they proposed a shared slack reclamation technique that shares the slack time among the processors. They have shown that the technique meets the deadline of all the tasks and reduces energy consumption as well. They used frame based tasks and assumed non-preemptive global scheduling.

Aydin and Yang (2003) proposed EDF based energy aware partitioned multi-processor scheduling algorithm called power partition that partitions the task set based on WFD decreasing heuristic. A new task assignment algorithm called RESERVATION is proposed that divides the processors and tasks into two categories: light and heavy. The light tasks are assigned to light processors and heavy tasks are assigned to heavy processors. The algorithm is a tradeoff between feasibility performance of First/Best Fit heuristics and the energy performance of WFD heuristic.

Chen et al. (2004) proposed a polynomial time approximation algorithm having approximation bounds of processor with task migration and processor speed constraint that minimizes the energy consumption. Chen et al. (2006) also proposed a technique that does multiprocessor scheduling in two phases: task remapping and slack reclamation. Task remapping decides whether the current task needs migration to another processor before proceeding with execution. In slack reclamation phase, slack time is used for slowing down the frequency.

Xian et al. (2007) presented an energy aware task partitioning and scheduling algorithm with uncertain workload under hard real time constraints. Task partitioning is achieved through better workload balancing by utilizing the probabilistic distributions of the task execution cycles.

Chen and Kuo (2007b) have done a survey of energy aware multiprocessor scheduling which covers aspects of energy optimization like DVFS and leakage power for input task sets composed of periodic tasks, aperiodic tasks, tasks with critical section etc. and derived the research gap. The research gaps presented by the authors are: scheduling on input task set containing periodic as well as aperiodic tasks, tasks in a task set having precedence constraint and application of these algorithms on heterogeneous platform.

Cong and Gururaj (2009) proposed an energy aware scheduling algorithm that considers resource constraints, precedence constraints among the tasks and input dependent variation in execution times of the tasks. The proposed algorithms are based on mathematical programming formulation of scheduling and voltage assignment.

Fujii et al. (2011) proposed an energy aware algorithm called Hetero Efficiency to Logical Processor (*HeLP*) for scheduling periodic tasks on prioritized SMT processors. Unlike other DVFS based processors, SMT processors cannot use *wcet* of task as these processors simultaneously execute instructions from multiple tasks. Author proposed *HeLP*, *HeLP* with temporal migration (HeLP-TM) and HeLP-TM-guarantee to reduce energy consumption while ensuring the real time constraints.

Moreno and Niz (2012) proposed DVFS based algorithm called Growing Minimum Frequency (GMF), for uniform multiprocessor systems on periodic task set and stated that their algorithm shows more energy saving than other algorithms on non-uniform multiprocessors.

2.4.3 Overall Energy Saving

Jha (2005) has done extensive survey of low power system scheduling and stated that research on uniprocessor low power scheduling is well established and there is a need to develop low power scheduling techniques for multi-processors. Since then the research focus switched to multiprocessor scheduling.

Langen and Juurlink (2006) designed a scheduling algorithm, LAMPS, for determining minimum number of processors which will consume minimum energy for a set of periodic tasks. This algorithm sets upper and lower bounds on number of processors and applies binary search to find minimum number of processors. It does not shutdown or procrastinate the job execution when the processor is idle. Langen and Juurlink (2009) extended their previous work by incorporating DVFS, shutdown and procrastination and found improvement in previous LAMPS algorithm. Chen et al. (2006) proposed a task assignment method in which Largest Task First with First Fit (LTF+FF) is used to allocate the tasks to processors. First the tasks are allocated according to LTF and then the processors with load less than critical speed are identified. The tasks on these lightly loaded processors are then re-assigned using FF in order to reduce the number of processors. Procrastination is also applied when the processors are idle for minimizing static energy consumption. The algorithm provides the approximation ratios of 1.283 and 2 when shutdown overheads are negligible and non-negligible respectively. Zeng (2009) proposed a DVFS based scheduling technique on periodic task sets which includes practical constraints such as discrete speed, idle power, inefficient speed and application specific power characteristics. Task assignment to processor is done using adaptive minimal bound first-fit (AMBFF) algorithm where it is stated that WFD based algorithms perform well when static energy consumption is ignored. AMBFF algorithm allocates the tasks to the processor with least frequency setting and also applies first fit heuristic in case of tie between two processors. This technique helps to reduce the tradeoff between dynamic and static energy consumption.

Yu et al. (2010) proposed a guided search heuristic to select best fit frequency level that maximizes the additional cycles of the adaptive task from a task graph. The algorithm also helps in selection of slack receiver task from a task graph. Bhatti et al. (2011) proposed a generic scheme called HyPowMan which makes use of a set of existing DVFS and DPM policies and applies a machine learning approach to dynamically select the best policy for any given workload on identical multi-processor systems.

Legout et al. (2014) proposed a solution to reduce the static energy consumption by efficiently using the low power states of multiprocessor systems. It works in two steps: offline step uses mixed integer linear programming to minimize number of preemptions and online step extends the existing scheduling algorithm to increase the length of idle period so that penalty incurred in switching to active mode is reduced.

Table 2.3 shows the summary of various energy efficient multiprocessor scheduling algorithms discussed in this section. It can be observed from table 2.3 that there is sufficient work done considering the periodic task model but no work has been done on mixed task model.

2.5 Scheduling Algorithms for Multi-core Processor Platform

For multi-core platforms, Yang et al. (2005) suggested a heuristic algorithm for scheduling frame based periodic tasks that share common deadline. This algorithm initially schedules the tasks at low frequency and keeps the idle cores in sleep state to reduce energy consumption. Later, it increases the frequency to meet the deadline of tasks. Isci et al. (2006) consider the energy optimization of multi-core processor using global power management technique on non-real time tasks model.

Seo et al. (2008) proposed two algorithms: Dynamic Repartitioning Algorithm and Dynamic Core scaling Algorithm. Former algorithm balances the task utilizations among the cores to minimize dynamic energy consumption while the latter algorithm adjusts the number of cores to reduce static energy consumption.

Table 2.3: Summary of Energy Efficient Multiprocessor Scheduling Algorithms

Broad Classification of Parameters	Parameters of Comparison Under Broad Category	References
Type of Energy	Dynamic Energy	Zhu et al., (2003), Aydin and Yang (2003), Chen et al., (2004), Wu et al., (2004), Chen et al., (2006), Xian et al., (2007), Fujii et al., (2011), Moreno and Niz (2012), Kuo. C (2014)
	Static Energy	Langen and Juurlink (2006), Legout et al., (2014)
	Dynamic Energy + Static Energy	Langen and Juurlink (2008), Chen et al., (2006), Zeng et al., (2009), Yu et al., (2010), Bhatti et al., (2011)
Task Allocation	Partitioned	Aydin and Yang (2003), Xian et al., (2007), Zeng et al., (2009), Fujii et al., (2011), Chen et al., (2006), Yu et al., (2010), Kuo. C (2014)
	Global	Zhu et al., (2003), Cong and Gururaj (2009), Moreno and Niz (2012), Legout et al., (2014)
	Hybrid	Chen et al., (2004), Chen et al., (2006)
Task Model	Periodic Task Model	Zhu et al., (2003), Aydin and Yang (2003), Chen et al., (2004), Wu et al., (2004), Chen et al., (2006), Xian et al., (2007), Cong and Gururaj (2009), Zeng et al., (2009), Fujii et al., (2011), Moreno and Niz (2012), Langen and Juurlink (2006), Langen and Juurlink (2008), Chen et al., (2006), Yu et al., (2010), Bhatti et al., (2011), Legout et al., (2014), Kuo. C (2014)
Method of Validation	Simulation Study on Synthetic Bench mark	Aydin and Yang (2003), Chen et al., (2004), Chen et al., (2006), Xian et al., (2007), Zeng et al., (2009), Fujii et al., (2011), Moreno and Niz (2012), Chen et al., (2006), Bhatti et al., (2011), Legout et al., (2014), Kuo. C (2014)
	Simulation Study on Real Bench mark	Langen and Juurlink (2006), Langen and Juurlink (2008),
	Simulation Study on Synthetic and Real Bench mark	Zhu et al., (2003), Cong and Gururaj (2009), Yu et al., (2010)

Lee (2009) proposed a DVFS based scheduling solution for periodic task model on lightly loaded multi-core processors. The technique exploits the overabundant cores by executing the tasks simultaneously and putting the core in sleep state if no task is assigned to it or if it is rarely used. Ren and Suda (2009) proposed load scheduling on multi-core and GPU platform for energy efficiency of multiplication of large matrices. The energy minimization is accomplished by multithreading CPU and parallel GPU by using parallel CUDA algorithm design. Lee et al. (2009) proposed two methods that reduce energy consumption of real time video streaming tasks. It considers the nonlinear scaling property of parallel execution speed up and finite discrete energy consumption rates of available frequencies.

Devdas and Aydin (2010) proposed two algorithms: Coordinated VFS and Coordinated VFS* on multi-core processors that have common clock. They applied global DVFS on periodic tasks and dynamically identified idle intervals to reduce both dynamic and static energy consumption.

Shieh and Chen (2010) proposed two energy aware algorithms for periodic tasks on dual core processors. One is offline approach in which they used integer linear programming (ILP) whereas the other one is online approach in which they proposed a heuristic algorithm. The results show that the online heuristic based algorithm is more energy efficient than the offline approach and is close to the optimal bounds of the ILP model.

Wei et al. (2010) considered scheduling multi-media real time tasks such as H.264 decoding using data-partitioning based approach that exploits parallelism over multi-core processors. This work reduces both dynamic and static energy using DVFS and DPM techniques. Task assignment is done using Largest Task First (LTF) technique. The experiments are done on AMD Phenon II platform using Linux Kernel 2.6.28.8. Modification to kernel is done to accomplish the work.

Xu et al. (2010) focus on energy consumption of processor cores and interconnection hardware. The computation and inter-core data transfer scheduling algorithm is proposed to reduce the interconnect energy consumption. Schonherr et al. (2010) have been motivated by Intel Turbo Boost Technology and suggested DVFS based scheduling technique for multi-core processors.

Huang et al. (2010) designed an algorithm to reduce static as well as dynamic energy consumption by applying DVFS and aggressive task reallocation strategies. The task reallocation algorithm analyzes run time idle intervals and takes aggressive reallocation decision which accumulates idle intervals and puts as many cores as possible in sleep mode.

Lu and Guo (2011) proposed two algorithms: pre-DVS and post-DVS which are based on fixed priority scheduling with task splitting. In post-DVS, DVFS is applied after scheduling while in pre-DVS, frequency selection of each task is done before scheduling according to the total utilization of task set and number of cores available.

Fu and Wang (2011) proposed a scheduling algorithm that combines the core level feedback in terms of current core utilization with processor level optimization to minimize dynamic and static energy consumption. The algorithm keeps track of current utilization of each core and dynamically responds to large variation in execution time by voltage and frequency scaling. Task consolidation is also carried out on a longer timescale in order to put the unused cores in shutdown mode which results in static energy saving. Independent periodic tasks are considered for scheduling. For experimental analysis, a scheduler in linux kernel is modified and the Mibench benchmark programs are used for testing the scheduler.

Khandhalu et al. (2011) proposed an energy efficient task assignment algorithm for voltage-island based multi-core processor. The algorithm considers fixed priority scheduling for periodic tasks on individual cores. Authors state that load balancing (which can be achieved by WFD) cannot give lower frequency when tasks are scheduled using Rate Monotonic Algorithm (RMA). Therefore, they proposed Single-clock domain multiprocessor Frequency Assignment Algorithm (SFAA) and have shown that it performs better than the WFD task allocation algorithm.

Lee (2012) suggested a heuristic scheduling algorithm on lightly loaded multi-core platform. The heuristic scheduling scheme schedules the periodic tasks in parallel on multiple overloaded cores to reduce dynamic energy and turns off the unused cores. He and Muller (2012b) investigated the problem of power optimization of hard real time systems on cluster based multi-core platform. They proposed a Simulated Annealing (SA) heuristic algorithm for task allocation. A cluster wide global frequency is selected at each scheduling point for each cluster. The jobs that are simultaneously running on all the cores in a cluster must scale to this frequency. Sheikh et al. (2012) have done a survey of allocation and scheduling algorithms, systems and software for reducing power and energy dissipation of single processor, multi-core processors and distributed systems.

He and Muller (2012a) offered a solution which combines DVFS and DPM for hard real time systems on clustered multi-core systems having multiple low power states with non-negligible switching overhead. The proposed algorithms use simulated annealing heuristic approach to minimize the overall energy consumption. He and Muller (2012b) have shown that energy efficient scheduling solution of uniprocessor cannot be directly applied to cluster based multi-core systems if DVFS and DPM state switching overheads are considered. They proposed a run time prediction technique that deals with DPM switching overhead and two solutions that improve the schedulability when DVFS switching overheads are considered.

Lin et al. (2012) focused on mitigating energy consumption and improving performance by developing a synchronization aware dynamic thread scheduling algorithm. This scheduling algorithm aims to reduce the busy waiting time of a task which is required to gain a lock. This paper considers spinlock for synchronization. By reducing the busy waiting time, it achieves less completion and turn-around time. As a result, it provides performance improvement and less energy consumption. The simulations are done on a simulator based on the scheduler in linux kernel 2.6 on a real bench mark application, i.e., digital video recorder.

Zhao et al. (2013) proposed a scheduling technique that can be applied on multi-core clusters on parallel applications with precedence constraint tasks. The dependency based task grouping method is designed to assign the parallel tasks to multiple cores. The algorithm achieves reduced energy consumption and improved resource utilization by assigning the tasks with highest dependency degrees to one core. The simulation study is carried out by using two real world applications viz. robot (Springer et al., 2006) and fppp (Kappiah et al., 2006).

Pagani and Chen (2013) suggested an algorithm namely Single Frequency Approximation (SFA) for minimizing energy consumption of a system that has multiple voltage islands. SFA is applied independently to each voltage island. The only work on mixed task set carried out by Ahmed et al. (2013) suggests a method of scheduling mixed task set for thermally constrained real time system.

Chen et al. (2013) state that integrating DVFS before DPM with scheduling may not give optimal solution. The proposed technique modeled the idle intervals of individual cores such that both DVFS and DPM can be optimized at the same time. The energy optimization problem is formulated as mixed integer linear programming problem.

Chen et al. (2014) proposed an energy efficient workload aware task scheduling (EEWA) algorithm to reduce dynamic energy consumption of multi-core processors. It is composed of two parts: workload aware frequency adjuster and preference based task stealing scheduler. The workload aware frequency adjuster uses DVFS to scale the frequency and voltage on the basis of the task's workload information collected from online profiling. The responsibility of balancing the workload among the cores and scheduling is effectively carried out by preference based task stealing scheduler. The task sets scheduled by EEWA scheduler are made from the parallel applications. Experiments are done by modifying the compiler and task scheduler of MIT Cilk (Frigo et al., 1998). The results state that the EEWA achieves up to 29.8% of energy saving with little degradation of CPU bound applications.

Lin et al. (2014) designed a scheduling algorithm that reduces energy consumption of multi-core processors by using DVFS on per core basis. The tasks are executed in two different modes: *batch mode,* which execute jobs in batches and *online mode,* in which the jobs have different timing constraints, arrival times and computational workloads. They proposed optimal scheduling policy known as Workload Based Greedy (WBG) policy to execute the jobs in batches. Another heuristic algorithm known as Least Marginal Cost is proposed for scheduling online jobs. Both the algorithms are theoretically proved and experimentally validated.

Zang and Chang (2014) have focused on optimizing the energy consumption in large scale enterprise data centers which execute multi-user applications on multi-core systems. The proposed algorithm is named as Cool scheduler that varies frequency and voltage by exploiting the run time program phases such as memory intensive phase and CPU intensive phase. It is built into the linux kernel and tested on SPEC CPU2006 and

Phoronix Test Suite on quad-core systems. In addition to reducing energy consumption, it reduces computation overhead and extra DVFS transitions.

Sheikh and Ahmad (2014) proposed efficient heuristic algorithms for joint optimization of performance, energy and temperature for task allocation on multi-core platform. They named this optimization problem as PETOS, Performance, Energy and temperature Optimization Scheduling. The authors state that the conventional multi-objective optimization approaches can be used to solve the problem but these methods take longer time and are not feasible for used at scheduling level. They proposed nine heuristics that are responsible for task assignment to multi-core processors and frequency selection decisions. The algorithms differ in a way they explore the scheduling decision space. The algorithms are classified into five categories: iterative scan method, adjust and schedule method, utility function based method, random and greedy method. It is found that the iterative scan and adjust and schedule methods provide most practical solutions. Greedy, random and utility function based methods achieve good diversity for large task graphs at the cost of huge execution time. The tradeoffs among the performance, energy and temperature at scheduling level are also discussed. In 2015, Sheikh and Ahmad proposed an evolutionary algorithm for solving PETOS problem in which they have integrated the strengths of multiple evolutionary algorithms and produced more accurate solution to PETOS problem. The authors extended their work on evolutionary algorithms for the energy optimization in their latest research in (Sheikh and Ahmad, 2016); Sheikh and Ahmad (2012) proposed a multi-objective evolutionary algorithm to PETOS (algorithm: SPEA).

Singh and Kaiser (2014) developed an energy efficient multi-core scheduler on the linux multi-core computing platform. The metric used to measure the performance/ energy efficiency is micro operations executed per joule (OPJ). As compared to completely fair scheduler that already exist in linux system, the energy efficient scheduling solution proposed in this work saves 30% energy.

Li and Wu (2014) proposed a DVFS based multi-core scheduling algorithm for scheduling arbitrarily arriving aperiodic tasks by using the sub-interval based scheduling method. In sub-interval based scheduling, the sub-intervals are identified dynamically on the basis of arrival time and desired execution requirement. The frequency selection is carried out on the basis of intensity of the sub-interval. At any scheduling time *t*, the greatest intensity among all the overlapped sub-intervals is considered to select the frequency of task execution. The algorithm optimizes dynamic as well as static energy consumption.

Lee et al. (2014) concentrate on reducing the chip level peak power consumption and temperature at design time. The chip level peak power consumption is another significant design parameter that determines the cost and size of chip and the underlying power supply. This energy efficient scheduling algorithm achieves the desired goal of energy saving without violating any real time constraints and it does not make use of any additional hardware for power management such as DVFS. The idea behind reducing the peak power is to restrict the concurrent execution of tasks on different cores which leads to peak power consumption.

Liu and Guo (2014) focused on relaxing the constraint of reducing the power consumption of individual cores and proposed a scheduling technique that reduces the energy consumption on voltage islands on multi-core platform. The motivation behind this work is that the modern multi-core processors adopt voltage islands in which a sub set of cores is grouped and named as island where each island is operated at a common frequency and voltage. The algorithm Voltage Island Largest Capacity First (VILCF) selects the frequency of task execution by utilizing the remaining capacity of the cores in the island efficiently without violating the real time constraints of the periodic tasks.

Islam and Lin (2014 and 2015) adopted learning based method for selection of DVFS technique and frequency scaling. The work in 2014 is applicable for uniprocessor which is extended for multi-core processors in their next work in 2015.

Fu et al. (2015) emphasized on the large amount of leakage power consumed by the main memory shared by DVFS enabled multiple cores in modern multi-core processors. The author designed an energy aware scheduling algorithm that optimizes dynamic energy by applying DVFS technique and static energy by putting shared main memory to sleep mode when all the cores have common idle time. The algorithm maximizes the common idle time in order to stretch the memory sleep time. The randomly generated periodic tasks are considered for scheduling. Another work by Fu et al. (2015b) concentrated more on maximizing the common idle time of all the cores to get larger memory sleep time. This algorithm is tested on synthetic and real benchmark suites.

Jin et al. (2015) considered Voltage Island based multi-core platform because many of the modern multi-core processors have adopted voltage island based hardware. The proposed energy optimization framework is capable of dynamically reconfiguring the voltage frequency island which requires less hardware cost and is suitable for diverse applications. It is also suitable for the multi-core systems with large number of cores.

Tran et al. (2016) proposed a scheduling solution to reduce the energy consumption of computational clusters with multi-core processors using DPM technique. In this work, the computation clusters are made up of heterogeneous commercial server. DPM is applied in a cluster where an idle server is turned-off and turn back to active state when it has a task to execute. The workloads used in the experiments are the real traces collected from production environment.

Sasaki et al. (2016) focused on reducing the energy consumed in three important shared resources: memory subsystem, CPU power and DRAM power. The proposed algorithm improves the performance by balancing the shared resource usage among the ready tasks. The evaluation results show that the algorithm shows better performance than the state of the art scheduling techniques which only considers memory subsystem contention.

Table 2.4 shows the summary of energy efficient multi-core scheduling algorithms. It can be observed from table 2.4 that researchers have addressed the issue of energy optimization at operating system level considering real time and non-real time tasks. Majority of the work focused on dynamic energy optimization and not on overall energy optimization which includes both dynamic and static energy optimization. In case of the existing work based on real time tasks, most of the energy efficient algorithms considered periodic task system and less attention is given to the mixed task systems.

Table 2.4: Summary of Energy Efficient Multi-core Scheduling Algorithms

Broad Classification of Parameters	**Parameters of Comparison Under Broad Category**	**References**
Type of Energy	Dynamic Energy	Chen et al., (2014), Lin et al., (2014), Zang and Chang (2014), Sheikh and Ahmad (2012, 2014, 2015, 2016), Singh and Kaiser (2014), Lee et al., (2014), Liu and Guo (2014), Jin et al., (2015), Islam and Lin (2015), Sasaki et al., (2016), Zhao et al., (2013), Lin et al., (2012), Khandalu et al., (2011), Shieh and Chen (2010), Xu et al., (2010) , Schonherr et al., (2010), Lee (2009), Ren and Suda (2009), Isci et al., (2006)
	Static Energy	Tran et al., (2016)
	Dynamic Energy + Static Energy	Li and Wu (2014), Fu et al., (2015), Fu and Wang (2011), Zhang et al., (2011), Wei et al., (2010), Devdas and Aydin (2010), Haung et al., (2010)
Task Model	Periodic Task Model	Lin et al., (2014), Liu and Guo (2014), Fu et al., (2015), Jin et al., (2015), Islam and Lin (2015), Fu and Wang (2011), Khandalu et al., (2011), Shieh and Chen (2010), Wei et al., (2010), Xu et al., (2010), Lee (2009)
	Aperiodic Task Model	Li and Wu (2014)
	Sporadic Task Model	Lee et al., (2014)
	Non-Real Time Task Model	Chen et al., (2014), Zang and Chang (2014), Sheikh and Ahmad (2012, 2014, 2015, 2016), Singh and Kaiser (2014), Tran et al., (2016), Sasaki et. al (2016), Lin et al., (2012), Schonherr et al., (2010), Ren and Suda (2009), Isci et al., (2006)
Method of Validation	Simulation Study on Synthetic Bench mark	Li and Wu (2014), Lee et al., (2014), Liu and Guo (2014), Jin et al., (2015), Islam and Lin (2015), Zhao et al., (2013), Zhang et al., (2011), Khandalu et al., (2011), Shieh and Chen (2010), Lee (2009)
	Simulation Study on Real Bench mark	Tran et al., (2016)
	Simulation Study on Synthetic and Real Bench mark	Sheikh and Ahmad (2012,2014, 2015, 2016), Fu et al., (2015), Zhao et al., (2013), Lin et al., (2012)
	Experimental Study	Chen et al., (2014), Zang and Chang (2014), Singh and Kaiser (2014), Sasaki et al., (2016), Fu and Wang (2011), Wei et al., (2010), Xu et al., (2010), Schonherr et al., (2010), Ren and Suda (2009), Isci et al., (2006)
	Simulation and Experimental Study	Lin et al., (2014)

2.6. Research Gaps and Challenges

From the literature described in this chapter, it can be observed that there is a need to address the issues of overall energy optimization by taking care of hard deadline periodic tasks and responsiveness of aperiodic tasks. Therefore, based on the survey of previous work in this area, the following gaps are identified:

1. There is a need to enhance the existing uniprocessor based energy efficient scheduling algorithms which can perform two important aspects:
    a. The energy efficient scheduling algorithms should be able to optimize dynamic as well as static energy consumption of the processor.
    b. The algorithms should be able to handle arbitrarily arriving aperiodic tasks in addition to periodic tasks. It should optimize the overall energy consumption without hampering the time constraints of periodic tasks and responsiveness of aperiodic tasks.
2. There is a need to optimize the overall energy consumption of multi-core processors as majority of the modern real time embedded systems make use of multi-core processors.
3. Very little work is done on energy optimization where the task model considered for scheduling is a mix of periodic and aperiodic tasks. Therefore, there is a need to design the energy efficient real time scheduling algorithms which can schedule mixed task sets and take care of the performance of both types of tasks along with energy optimization.
4. In case of multi-core scheduling, majority of the research work focused either on partitioned approach or on global approach. There is a need to combine both the approaches to overcome the limitations of both of approaches.

Table 2.5 shows the observations from the previous literature and highlights the research gaps that are targeted in this research work.

Table 2.5: Observations and Research Gaps

Processor Platform	**Task Model**	**Dynamic Energy Optimization**	**Overall Energy Optimization**
Uniprocessor	Periodic Task Model	Sufficiently Addressed	Sufficiently Addressed
	Mixed Task Model	**Sufficiently Addressed**	**Not Sufficiently Addressed**
Multiprocessor	Periodic Task Model	Sufficiently Addressed	Sufficiently Addressed
	Mixed Task Model	Not Addressed	Not Addressed
Multi-core Processor	Periodic Task Model	**Not Sufficiently Addressed**	**Not Sufficiently Addressed**
	Mixed Task Model	**Not Addressed**	**Not Addressed**

In brief, there is a need to design a complete scheduling framework which can schedule mixed task sets and minimize overall energy consumption of a DVFS enabled multi-core processor having individual clock domain. The proposed energy efficient scheduling algorithms take care of the hard deadline periodic tasks and responsiveness of aperiodic tasks.

Chapter 3

Dynamic Voltage and Frequency Scaling Based Multi-Core Scheduling

This chapter discusses the design and development of the proposed energy efficient real time scheduling algorithms those are capable of reducing dynamic energy consumption on uniprocessor and homogeneous multicore platforms.

3.1 Introduction

Energy optimization is an important concern in contemporary real time embedded systems based around microcontrollers as well as high performance microprocessors. Specifically, the dynamic energy consumed by the hardware components within the microprocessor contributes a large amount to the overall energy consumption. Dynamic energy consumption is caused due to the presence of switching capacitance in the processor that comes into play when the processor is in running state. The dynamic energy is consumed by clock circuit, datapath, memory, control and input/output devices present on the processor. In Tiwari et al. (1998), the authors stated that the major contributor of dynamic power is the clock circuitry as it is active for the entire time duration. This circuitry includes clock generator, clock driver, clock distributor, latches and clock loading circuits. The datapath is the next largest consumer of energy. Memory, control circuit and I/O devices consume relatively less amount of energy as compared to clock circuitry and datapath.

The issue of large amount of energy consumption can be addressed at various levels of abstraction resulting in two broad categories: hardware based techniques and software based techniques. The hardware based techniques include clock gating, power gating, transistor sizing, low power logic synthesis etc (Benini et al., 1994; Tiwari et al., 1996; Borah et al., 1996; Macii et al., 2008; Li et al., 2013).

Clock gating is a most effective technique to optimize dynamic energy consumption, since the clock distribution network consumes 40% of the total energy budget of a CMOS circuit and it operates at highest switching frequency compared to any other signal driving a large amount of capacitive load (Macii et al., 2008; Benini et al., 1994). It dynamically identifies the unused hardware components and disables the clock to these components, so that these components get deactivated and stop consuming energy. Another hardware based technique called power gating is effective in reducing the leakage power, in which sleep transistors are placed between the logic circuits and the ground rail (Roy et al., 2003). There exist techniques that integrate both clock and power gating to mitigate the overall energy consumption. There are many works in integration of clock and power gating techniques (Macii et al., 2008; Li et al., 2013). Transistor sizing is another effective technique that reduces the dynamic energy consumption by increasing/reducing the channel width of a transistor. By increasing the channel width, the current drive capability of the transistor increases there by reducing the signal rise/fall time at the gate output. This results in reduction in dynamic energy consumption (Borah et al., 1996; Raja et al., 2006).

As the ultimate job of a processor is to execute software, the same should be designed in a way that the processor consumes less energy. This motivated researchers to develop software based energy optimization techniques. These techniques can be implemented by using compiler optimization, instruction level optimization, source code optimization and by modifying the scheduler in operating system (Hsu and Kremer, 2003; Tiwari et al., 1996; Pillai and Shin, 2001).

One of the popular software based techniques is DVFS which can be applied on processors that are capable of varying frequency and supply voltage dynamically. It exploits the fact that in majority of the applications, high performance (or high frequency/speed) is required for only a small fraction of time while most of the time low performance (or low frequency/speed) is sufficient. DVFS technique works on the idea of dynamically adjusting operating frequency and supply voltage of the processor based on its current load. Typically, a processor can operate at low supply voltage at a lower execution frequency whereas it operates on high supply voltage at higher execution frequency. Executing a task at a lower frequency is possible because of the fact that the *aet* is less than or equal to *wcet* of a task. The difference between *wcet* and *aet* is called slack time. This available slack time is used to slow down the processor to save energy. Since energy consumed per cycle with CMOS circuitry scales quadratically with the supply voltage, DVFS potentially reduces the processor energy consumption through frequency and voltage scaling. In order to decide when to execute at higher or lower frequency, DVFS technique requires the cooperation of operating system scheduler. So DVFS based techniques are always applied in coordination with operating system schedulers.

This chapter discusses the design and analysis of the following proposed scheduling algorithms: energy efficient scheduling algorithm on uniprocessor platform using two different aperiodic servers (EEDVFS and EE-UCS), non energy aware Multi-Core Scheduler (non-DVFS MCS) and energy aware Multi-Core Scheduler (MCS). The proposed scheduling algorithms are designed for scheduling real time task sets that contain a mix of periodic and aperiodic tasks. The energy efficient uniprocessor based scheduling algorithm is designed using two different aperiodic servers, Deferrable Server (DS) and Total Bandwidth Server (TBS). The non-DVFS MCS schedules the mixed task set on multiple homogeneous cores without optimizing energy consumption. MCS schedules the mixed task set on multiple homogeneous cores and optimizes the dynamic energy consumption using DVFS energy optimization technique.

3.2 System Model

This section describes the system model assumed for designing the proposed scheduling policies in this chapter. The system model includes energy model that explains the type of energy considered for optimization, processor platform that is used as a hardware platform and the task model that is being scheduled by the scheduling policies.

3.2.1 Energy Model

The total power consumed by any processor comprises of three components: dynamic power (P_{AC}), static power (P_{DC}) and power required to keep the processor ON (P_{ON}).

The dynamic power is consumed when processor is running at certain frequency. In CMOS based processors, dynamic power and maximum frequency can be defined by equation 3.1 and 3.2 (Jejurikar et al., 2004):

$$P_{AC} = V_{dd}^2 \times f_{CLK} \times C_{EFF} \tag{3.1}$$

$$f_{CLK} = k \times \frac{(V_{dd} - V_t)^2}{V_{dd}} \tag{3.2}$$

Where V_{dd} is the supply voltage, f_{CLK} is the clock frequency, V_t is threshold voltage, k is a constant and C_{EFF} is the effective switching capacitance. The static power consumption is caused due to sub-threshold leakage current (I_{subn}) and reverse bias junction current (I_j). It can be represented in terms of I_{subn} , I_j and body bias voltage (V_{bs}) by equation 3.3 (Jejurikar et al., 2004):

$$P_{DC} = V_{dd} \times I_{subn} + |V_{bs}| \times I_j \tag{3.3}$$

In addition to dynamic and static power, there is an implicit consumption of power when the processor is in idle state. This power is denoted by P_{ON} and is required to keep the processor on. It is caused mainly due to I/O sub-systems, PLL circuits etc. The contributors of P_{ON} depend on technology and architecture of the processor.

The total power (P) is calculated by summing $P_{AC,}$ P_{DC} and P_{ON} and is given by the following equation:

$$P = P_{AC} + P_{DC} + P_{ON} \tag{3.4}$$

Total energy consumption per cycle, E, is given by -

$$E = E_{AC} + E_{DC} + E_{ON} \tag{3.5}$$

Where,

$$E_{AC} = P_{AC} \times f_{CLK}^{-1} = V_{dd}^2 \times C_{EFF} \tag{3.6}$$

$$E_{DC} = P_{DC} \times f_{CLK}^{-1} = \left(V_{dd} \times I_{subn} + |V_{bs}| \times I_j\right) \times f_{CLK}^{-1} \tag{3.7}$$

$$E_{ON} = P_{ON} \times f_{CLK}^{-1} \tag{3.8}$$

Where E_{AC}, E_{DC}, E_{ON} are dynamic energy, static energy and energy required to keep the processor ON respectively.

Equation 3.2 states that in modern CMOS processor, frequency is directly proportional to supply voltage. According to equation 3.6 and 3.7, it is seen that there is quadratic and linear dependence of supply voltage on dynamic and static energy respectively. Therefore, from equations 3.2 and 3.6, it can be shown that dynamic energy consumption is directly proportional to the square of clock frequency f_{CLK} (or from equations 3.1 and 3.2, dynamic power consumption is directly proportional to cube of clock frequency f_{CLK}). This means that if frequency is increased to improve the performance, it results in increased energy consumption. Thus there is a tradeoff between processor performance and energy consumption. DVFS techniques are capable of dealing with this tradeoff.

Even though it is observed that there is quadratic and linear dependence of dynamic and static energy respectively on supply voltage, decreasing the voltage beyond a certain limit does not result in energy saving. This is because, the static energy consumption increases if the processor speed decreases below a threshold. This threshold is called critical speed. If the processor runs below this speed, the static power consumed due to leakage current nullifies the gains of DVFS. Therefore, in order to retain the benefits of DVFS, frequency scaling should be limited to critical speed (Jejurikar et al., 2004).

3.2.2 Processor and Task Model

A homogeneous multi-core processor consists of M identical processor cores which are denoted as $\{C_0, C_1, \ldots\ldots, C_{M-1}\}$. Each core is capable of changing frequency and voltage dynamically and has its own physical clock. It is assumed that the resource sharing among the cores do not incur any inconsistency between the tasks executing across the cores.

The target task set includes a mix of periodic and aperiodic tasks with hard and soft deadlines respectively. The periodic tasks considered in this work are highly critical in nature such that they cannot miss the deadline. The examples of highly critical periodic tasks can be target sensing and track correction in a missile guidance system. In a guided missile, a computer is mounted on the missile which periodically senses the target and corrects the path/track. If these tasks are not completed by a deadline or before, the missile may home onto the unwanted area which may even cause a disaster. On the other hand, aperiodic tasks do not have hard timing constraints but they require a good average response time. A poor response time degrades the system performance. An example of an aperiodic task is changing from manual to auto pilot mode or vice-versa in a computer on-board an aircraft.

The tasks (periodic/aperiodic) are independent and preemptive in nature. A task set T consists of N periodic tasks and K aperiodic tasks. Each periodic task P_i is defined by $\{a_i, c_i, p_i, D_i\}$ and each aperiodic task A_i is defined by $\{a_i, c_i\}$, where a_i is the arrival time, c_i is the *wcet* at maximum frequency, p_i is the period or inter-release time and D_i is the deadline. Periodic tasks have implicit deadlines ($\forall\ P_i$, $p_i = D_i$) and are in phase ($\forall\ P_i$, $a_i = 0$). Utilization of periodic tasks in a task set can be defined as $U_p = \sum_{i=1}^{N} c_i/p_i$. Aperiodic tasks arrive arbitrarily at any time instance during the schedule. They can be normally scheduled using any of the standard bandwidth preserving scheduling algorithms.

3.3 Scheduling on Uniprocessor Systems

In order to design an energy efficient scheduling algorithm for mixed task system on uniprocessor platform, it is required to meet three major objectives. The first objective is to maintain the schedulability of periodic tasks in presence of aperiodic tasks so that the aperiodic tasks should not prevent the periodic tasks from finishing before the deadlines. The second objective is to serve the aperiodic tasks with reasonable average response time. For achieving the responsiveness of aperiodic tasks, many bandwidth preserving scheduling algorithms such as Deferrable Server (DS), Sporadic Server (SS), Total Bandwidth Server (TBS), Constant Bandwidth Server (CBS) etc are available. The third objective is to optimize the processor energy consumption while scheduling a mixed task system.

The basic idea behind any bandwidth preserving scheduling algorithm is to reserve some portion of the processor utilization for the execution of aperiodic tasks. The above mentioned aperiodic servers differ in a way they utilize the preserved processor utilization (or bandwidth) for the execution of aperiodic tasks. The responsiveness of aperiodic tasks depends on the way the reserved bandwidth is utilized by the aperiodic tasks.

The proposed algorithms presented in this work are implemented using two different aperiodic servers: DS and TBS. The brief description of DS and TBS is given below (Liu, 2008):

- **Deferrable Server (DS):** The deferrable server is one of the simplest bandwidth preserving servers. In this algorithm, an aperiodic server with fixed period p_s and execution budget e_s is maintained for scheduling arbitrarily arriving aperiodic tasks. This aperiodic server is scheduled along with periodic tasks with additional consumption and replenishment rules for aperiodic tasks. The consumption and replenishment rules are defined as follows:

- *Consumption Rule:* The execution budget of the server is consumed at the rate of one per unit time whenever the server executes.
- *Replenishment Rule:* The execution budget is replenished to e_s after every regular interval of period p_s. The budget is not allowed to be accumalated from period to period.

The aperiodic task is not allowed to execute more than e_s units of time in one server period. It gets preempted if the budget is exhausted and again scheduled in next interval when the budget is repleneshed. This results in late response time of aperiodic tasks.

Total Bandwidth Server (TBS): TBS reserves the remaining utilization of the core other than periodic utilization which is named as server utilization U_s. Aperiodic jobs cannot execute beyond server budget U_s. In other words, utilization of aperiodic tasks cannot exceed U_s on each processor core. Therefore, on a processor core where a mix of periodic and aperiodic tasks are executed, periodic tasks will not miss their deadlines, if and only if

$$U_p + U_s \leq 1 \tag{3.9}$$

The replenishment rules of TBS are improved as compared to the rules of DS. In this case, unlike DS, aperiodic tasks can utilize the server budget at a stretch. There is no need to wait till next interval for replenishment. The replenishment rules of TBS with execution budget e_s, server utilization are u_s, virtual deadline of aperiodic job d and current time t are as follows:

- Initially, $e_s = 0$ and $d = 0$.
- When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue, set d to max $(d,t) + e/u_s$ and $e_s = e$.
- When the server completes the current aperiodic job, the job is removed from its queue.
    - If the server is backlogged, the server deadline is set to $d + e/u_s$ and $e_s = e$.

- If the server is idle, do nothing.

3.4 Proposed Energy Efficient Uniprocessor based Scheduling Algorithm (EEDVFS)

This section describes an energy efficient uniprocessor based scheduling algorithm for scheduling mixed task sets. An energy efficient dynamic voltage and frequency scaling (EEDVFS) scheduling algorithm is designed for scheduling mixed task set and optimizing energy consumption of uniprocessor platform. In this algorithm, the periodic tasks are scheduled using EDF and aperiodic tasks are scheduled using DS. A DVFS technique called utilization update (UU) (Kim et al., 2002) is applied to scale the execution frequency of periodic tasks in a task set. The aperiodic tasks are executed at maximum voltage and frequency in order to achieve better response time.

3.4.1 Notations

Periodic task set T = $\{T_0, ...,T_{n-1}\}$ represents *n* periodic tasks. Each periodic task T_i is represented by $\{\Phi_i, P_i, C_i, D_i\}$ where Φ, P, C and D represent phase, period, *wcet* and deadline respectively and E_i represents *aet* of periodic task. Aperiodic task set A = $\{A_0,.....,A_{m-1}\}$ represents *m* aperiodic tasks. Each aperiodic task A_i is represented by $\{A_i, C_i\}$ where A and C represent arrival time and *wcet* respectively. $T_{ds} = \{P_{ds}, C_{ds}\}$ represents DS with period P_{ds} and execution budget C_{ds}. The parameters PRQ and ARQ represent periodic and aperiodic queues respectively and are initialized to NULL. The current time is represented by t and is initialized to 0. The parameters U_i, U_{ds} and UT represent utilization of periodic job, utilization of aperiodic job and total utilization respectively. $T_{<i,j>}$ denotes j^{th} job of i^{th} task in the task set.

The parameters f_{max} and v_{max} represent maximum frequency and voltage respectively, f_{opt} and v_{opt} represent optimum frequency and voltage respectively. The parameter n_d_p is next decision point, RET is Remaining Execution Time, A_{np} and A_{na} represent arrival time of next periodic and aperiodic job respectively. J_{hp} and J_{ha} represent job at head of PRQ and ARQ respectively. The parameter t_j represents number of jobs in PRQ at time t.

3.4.2 EEDVFS Algorithm

Algorithms 3.1 and 3.2 show the scheduling algorithm EEDVFS and its corresponding frequency selection algorithm respectively. The periodic tasks are ordered on the basis of EDF scheduling policy. At each scheduling point, if aperiodic job queue is empty or server budget is exhausted, then the periodic job is scheduled and executed at the scaled frequency. In presence of aperiodic job, if the server budget is available, the aperiodic job executes at maximum frequency. If the remaining execution time of the aperiodic job is greater than the server budget then the aperiodic job is preempted and it resumes after replenishment of the budget. If the periodic job queue is empty and server budget is exhausted or there is no aperiodic job ready for execution, then the processor remains idle. The frequency scaling is done on the basis of current utilization of the processor at every scheduling point. The current processor utilization is calculated on the basis of the *aet* of the completed jobs and *wcet* of the newly arrived or preempted jobs. The algorithm 3.2 shows the frequency scaling technique used in EEDVFS. In this algorithm, the aperiodic job is executed in chunks whenever the server budget is available. The DS based aperiodic task scheduling leads to longer response time of the aperiodic jobs as compared to other aperiodic servers such as CBS, TBS etc. The working of EEDVFS algorithm is illustrated through the following example task set and the corresponding schedule.

Tasks	**Arrival**	**Period**	***wcet***	**Deadline**	**# of jobs in a HP**	***aet* of jobs**
T_{ds}	0	10	3	10	-	-
T_0	0	25	7	25	2	3, 4
T_1	0	50	15	50	1	8
Ap_0	5	-	5	-	-	5
Ap_1	10	-	5	-	-	5

Time	R	C	P (RET)	U(R/C)	f_{opt}	*aet*
0	$J_{<0,0>}$,$J_{<1,0>}$	-	-	0.58	0.75	$J_{<0,0>} = 4$
4	-	$J_{<0,0>}$	-	0.42	0.5	$J_{<1,0>} = 16$
5	A_{p0}	-	$J_{<1,0>}$ (7.5)	-	1	$A_{p0} = 5$
8	-	-	A_{p0} (2)	-	0.5	$J_{<1,0>} = 15$
10	A_{p1}	-	$J_{<1,0>}$ (6.5)	-	1	$A_{p0} = 2$
12	-	A_{p0}	-	-	1	$A_{p1} = 5$
13	-	-	A_{p1} (4)	-	0.5	$J_{<1,0>} = 13$
20	-	-	$J_{<1,0>}$ (3)	-	1	$A_{p1} = 4$
23	-	-	A_{p1} (1)	-	0.5	$J_{<1,0>} = 6$
25	$J_{<0,1>}$	-	-	0.88	1	$J_{<1,0>} = 2$
27	-	$J_{<1,0>}$	-	0.74	0.75	$J_{<0,1>} = 5.33$
30	-	-	$J_{<0,1>}$ (1.75)	-	1	$A_{p1} = 1$
31	-	A_{p1}	-		0.75	$J_{<0,1>} = 2.33$
33.33	-	$J_{<0,1>}$	-	-	-	-

Where R-Release, C-Completion, P(RET) - Preemption (Remaining Execution Time of preempted job at fmax), U(R/C) - Total Utilization upon release or upon completion, f_{opt} - Optimal Frequency, *aet* - Actual execution time of executing job.

Algorithm 3.1: Proposed EEDVFS Algorithm

```
EEDVFS (T, A, PRQ, ARQ, t)
Begin
        Join all pending periodic jobs till t to PRQ
        Join all pending aperiodic jobs till t to ARQ
        if (ARQ(!empty) and ($C_{ds}$>0)) then
                n_d_p ← t + min ($C_{ds}$, RET($J_{ha}$))
                execute $J_{ha}$ till n_d_p at $f_{max}$ and $v_{max}$
                $C_{ds}$← n_d_p - (min($C_{ds}$, CT($J_{ha}$)) + t )
                t ← n_d_p
        else if(PRQ (!empty)) then
                n_d_p ← min(t + RET($J_{hp}$), $A_{na}$, $R_p$, $A_{np}$)
                execute $J_{hp}$ till n_d_p at $f_{opt}$ and $v_{opt}$
                t ← n_d_p
        else if((ARQ(empty) or $C_{ds}$ = 0) and PRQ(empty)) then
                CPU is idle till any other periodic or aperiodic job arrives or server
                 replenishes the execution budget
End
```

Algorithm 3.2: Frequency Selection of Proposed EEDVFS Algorithm

```
Select_Frequency ($J_k$)          // $J_k$ is periodic job
Begin
        if ($J_k$ Released) then
                set $U_k \leftarrow C_k/P_k$    where $0 \leq k < p$
        else if ($J_k$ Completed) then
        set $U_k \leftarrow E_k/P_k$ where $0 \leq k < p$
        set $U_{ds} \leftarrow C_{ds}/P_{ds}$
        $UT \leftarrow \sum_{i=0}^{t_j} U_i + U_{ds}$
        Select frequency $f_{opt} \in \{f_1, f_2,..f_{max} \mid f_1 < f_2 <...< f_{max}\}$ such that $UT \leq f_{opt}/f_{max}$
        return $f_{opt}$
End
```

3.5 Scheduling on Multi-core Systems

Task scheduling on multiprocessor/multi-core systems is comparatively difficult than scheduling on uniprocessor systems. Optimal schedulers are available for scheduling independent real time tasks on uniprocessors. These schedulers have polynomial time complexity. However, scheduling independent real time tasks on multiprocessor/multi-core processor is an NP-Hard problem. Scheduling of real time tasks on multi-core systems consists of two sub-problems: task allocation to processors and scheduling the tasks on individual processor cores.

3.5.1 Task Allocation

The task allocation problem deals with allocation of tasks to different processor cores. This can be done statically or dynamically. The static task allocation is known as partitioned approach whereas the dynamic task allocation is known as global approach.

3.5.1.1 Partitioned Task Allocation Strategy

In the partitioned approach, once a task is allocated to a core, it permanently resides on that core and all the jobs corresponding to this task are scheduled on the assigned core. This approach requires prior knowledge about the attributes of the tasks. The major advantage of this approach is that the well-established uniprocessor scheduling algorithms can be directly applied to schedule the sub set of tasks on each core. The majority of the hard real time systems follow this approach (Davis and Burns, 2011). Partitioned approach for task allocation is known to be NP - hard problem. Various partition based approximation algorithms exist which give polynomial time solutions (Burchard et al., 1995; Anderson et al., 2001; Anderson and Jonsson, 2003; Lopez et al., 2004; Zapata and Alvarez, 2004). However, these algorithms waste processing capacity of the multiprocessor as compared to the optimal approach.

Utilization balancing and bin packing algorithms are the well known partition based approximation algorithms available. At each core, dynamic priority scheduling algorithm like EDF is used for scheduling (Davis and Burns, 2011). These algorithms are discussed in brief below:

A. ***Utilization balancing algorithm***

Utilization balancing algorithm orders the tasks based on their increasing utilization and inserts the ordered tasks in a queue. It then removes the highest priority task (task with highest utilization) from the front of the queue and assigns it to the least utilized processor core. This results in balancing the utilization of tasks among the cores. However, it is difficult to achieve perfectly balanced load on each core using this algorithm as it gives suboptimal results. This algorithm is suitable when the processor cores in use at any given time are fixed (Krishna and Shin, 1997).

B. Bin Packing Algorithm

Bin packing is a standard heuristic that is used to solve the problem of task allocation in multiprocessor scheduling theory. The bins are considered as processors and the tasks are the objects that are required to be packed in respective bins. The number of objects packed in a bin should not exceed certain bound. For EDF scheduling policy, the utilization bound on each processor core is 1 as the schedulability of EDF is 1. So the bin packing algorithms used for task allocation make use of utilization bound of 1 when scheduling tasks using EDF scheduling policy.

There exist several bin packing algorithms in the literature (Garey and Johnson, 1979). According to Lopez et al. (2004), they can be categorized into two types: Reasonable Allocation (RA) and Reasonable Allocation Decreasing (RAD). A reasonable allocation algorithm is defined as the one that fails to allocate a task to a multiprocessor only when the task does not fit into any processor. The reasonable allocation decreasing algorithms are similar to *RA* algorithms with only one difference. That is, before allocation, the tasks are ordered based on non-increasing order of utilizations and are allocated to the processors sequentially in that order. The bin packing heuristic algorithms are classified into two categories as below:

- *RA* Algorithms: First Fit (FF), Best Fit (BF), Next Fit (NF), Worst Fit (WF)
- *RAD* Algorithms: Worst Fit Decreasing (WFD), First Fit Decreasing (FFD), Best Fit Decreasing (BFD)

First Fit algorithm allocates tasks to a processor core with lowest index such that the sum of its current utilization and utilization of newly arrived task does not exceed the utilization bound of the core. If the sum exceeds the bound, then the new task is not allocated to the core. The algorithm then finds the next lower indexed core which satisfies the bound condition. Similarly, the *Best Fit* algorithm finds a core with smallest available utilization which can be allocated to the new task. If more than one core is available, then it assigns the task to the core with lower index. In *Next Fit* algorithm, after allocating the new task to the current core, the next new task will be allocated to the same core only if the utilization bound of that core does not exceed the maximum capacity otherwise it is allocated to the next core. Lastly, *Worst Fit* allocates a new task to a core that has maximum available utilization.

Application of *RAD* algorithms is possible only if all the tasks in a task set are available before allocation. This is because the tasks are required to be ordered before allocation. These algorithms are more suitable for hard real time tasks as the attributes are typically known a priori. Davis and Burns (2011), in a survey, have stated that, for implicit deadline periodic task sets, the largest worst case utilization bound for any partitioning algorithm is given by:

$$U_{wc} = (m+1)/2 \tag{3.10}$$

Where m is the number of processors and U_{wc} is the worst case utilization of the system. The above equation 3.10 holds true because it is not possible to schedule $m+1$ tasks with execution time $1+\varepsilon$ each and a period of 2 on m processor cores. This further shows that if all the tasks in a task set have utilization greater than 50%, then they cannot be partitioned amongst the cores. This results in waste of processor capacity. This is the main disadvantage of partitioned approach. The global approach explained below does not suffer from this limitation.

3.5.1.2 Global Task Allocation Strategy

In case of global task allocation, the jobs that are ready for execution are dynamically assigned to the processor cores. In order to make this possible, all the ready jobs are placed in a common priority queue (also known as global queue) and dispatched to the processor cores for execution as soon as processor cores become available. It is also possible to migrate a running job from one core to another if a higher priority job arrives on the current processor core. This method does not require the entire task set to be available before allocation. This approach is suitable for scheduling the tasks that arrive arbitrarily at run time. Thus, this method is suitable when aperiodic or sporadic tasks are present in the system as they arrive arbitrarily at run time. In this approach, since the allocation is done on the basis of current load of the individual core, the schedulable core utilization is better than the utilization achieved in static approach.

The two popular global task allocation algorithms usually applied in distributed systems are *focused addressing and bidding* and *buddy algorithm* (Krishna and Shin, 1997). In focused addressing and bidding algorithm, each processor maintains two tables: *status table* and *system load table*. Status table contains the information about the tasks that are ready for execution and system load table contains the current load information of each of the processors in the system. Each of the processors broadcast their available utilization, so that all the other processors can update their system load tables. Whenever a task arrives at a processor, the processor checks its system load table and decides whether to schedule that task or migrate it to some other processor. In case of migration, to select a processor, it identifies some lightly loaded processors that can accommodate this task by sending a request for bid to these processors. The task is then migrated to one of the least loaded processors. The issue that may arise in this technique is that the system load table may contain obsolete information. This technique may also have high communication overhead since every processor has to communicate its current load to other processors at regular intervals. Further, selecting a processor for task allocation also requires message exchange. The *Buddy algorithm* performs better than focused addressing and bidding in a way that it categorizes the processors into two types: under loaded and overloaded based on a threshold value, so that the candidate processors are reduced. Also, the processor does not broadcast their current load periodically, rather they broadcast only when the status of processor changes from under loaded to overloaded or vice versa. The communication overhead in this case is relatively less than the previous method due to small number of message passing between the processors.

3.5.1.3 Hybrid Task Allocation Strategy

The hybrid approach (Davis and Burns, 2011) exploits the advantages of both partitioned and global approaches and overcomes their limitations. It tries to reduce the fragmentation of available processor capacity and also tries to increase the maximum utilization bound of 50% that exists in partitioned approach. It also tries to reduce the global queue length and potentially reduce the migration overheads that exist in global approach. In order to achieve this, there exist two techniques: semi-partitioned approach and clustering.

In semi-partitioned approach, one or few tasks from a task set are split into sub tasks. These sub tasks are assigned globally to two or more processors and rest of the tasks are partitioned. With this approach, utilization bound of partitioned approach is improved and preemptions are also reduced due to restricted global assignment.

In clustering, the processors are grouped into clusters and task allocation is done among the clusters. Each cluster may share the same cache. This reduces the penalty for migration. This also reduces the length of global queue as each cluster would have a separate queue that potentially reduces the migration overhead.

3.5.2 Task Scheduling

The second step in multiprocessor scheduling is task scheduling. The multiprocessor scheduling using partitioned approach takes advantage of optimality results of the uniprocessor scheduling since after partitioning, the tasks allocated to each core are scheduled on respective cores. The jobs of the tasks belonging to a core are maintained in separate run queues designated to that core (Davis and Burns, 2011). Rate monotonic (RM) algorithm is the fixed task priority optimal preemptive uniprocessor scheduling algorithm for periodic and sporadic task systems with implicit deadlines (Liu and Layland, 1973). Similarly, deadline monotonic (DM) is the optimal preemptive scheduling algorithm for tasksets with constraint deadlines while it is not optimal for tasksets with arbitrary deadlines (Leung and Whitehead, 1982). EDF is the job level fixed priority preemptive scheduling algorithm for periodic and sporadic tasks independent of the deadline constraint (Dertouzos, 1974).

In case of global scheduling, assigning a task depends on the state of all the processors. The tasks are permitted to migrate from one core to another. The ready jobs are maintained in a single run queue. This approach permits job-level migration in which a particular job executes on a respective core but jobs of the same task can execute on different cores. There exist many fixed job priority global scheduling algorithms for periodic tasksets with implicit and constraint deadlines (Davis and Burns, 2011).

Both the scheduling approaches have advantages and disadvantages over the other one (Davis and Burns, 2011). Partitioned scheduling has the following advantages and disadvantages as compared to global scheduling:

- There is no penalty of migration cost as the tasks run on respective cores.
- If a task overruns its worst case execution budget, then it only affects the other tasks on the same core.
- It is easy to maintain a separate run queue per core as compared to single run queue in global approach.
- The major disadvantage of this scheduling approach is that the available processing capacity becomes fragmented resulting in large amount of capacity unused by the tasks.
- Another disadvantage is that the partitioning problem is NP-Hard.

  Global scheduling has the following advantages and disadvantages as compared to partitioned scheduling:

- The context switching and preemptions are relatively fewer in this approach as the scheduler will only preempt a task when there is not a single idle processor core.
- The slack time generated by a task can be utilized by all the tasks in the task set and not just by the subset of tasks in the task set.
- If the task over runs its worst case execution budget, the probability of the deadline miss is less because of the availability of multiple cores.

- The major disadvantage of global scheduling is the overhead in terms of communication load and cache misses that is incurred due to migration of jobs. Thus it is more advantageous to use a hybrid scheduling approach which takes advantages of both partitioned and global scheduling approaches.

3.6 Proposed Multi-core Scheduling Algorithm (MCS)

This section describes the design of a proposed energy efficient real time scheduling algorithm, Multi-Core Scheduler (MCS), for mixed task set on multi-core processor platform. The algorithm is explained with the help of an example task set scheduled on a dual core processor.

3.6.1 Proposed Algorithm

The problem of dynamic energy optimization using DVFS on multi-core platform for mixed task model comprises of three parts: allocation of tasks to the cores, assignment of frequency to each job and scheduling of jobs at each scheduling point.

In order to ensure the hard deadlines of periodic tasks are met, these tasks are statically partitioned among M processor cores. Once periodic tasks are allocated to the respective cores, the jobs corresponding to these tasks execute on the allocated core. Because of highly critical nature of periodic tasks, they are not allowed to migrate to other core even if the cores are homogeneous. The reason behind this restriction is that migration requires preemption of task on one core and resumption on another core which in turn requires restoring the context on the other core. This can result in cache misses and miss penalty. This kind of migration overhead may cause deadline miss of periodic jobs especially in overloaded systems. On the other hand, aperiodic tasks are assigned to one of the processor cores dynamically upon their arrival. They are allowed to migrate to other cores upon preemption. This is because, there is no hard deadline requirement and migration to other cores may provide early completion of aperiodic jobs in lightly loaded systems. The selection of a core for an aperiodic task execution is based on ensuring early response time.

At each scheduling point, if a scheduled job is periodic, a scaled frequency is assigned to the job which results in reduction of dynamic energy consumption. If the job is aperiodic then maximum frequency is assigned which improves its response time.

Selection of frequency is discussed later in this section. The scheduling policy used for periodic tasks is EDF since it is an optimal scheduling algorithm for uniprocessor platform and TBS is used to schedule aperiodic tasks because it is simple and provides good response time over other aperiodic servers (Spuri and Buttazzo, 1996).

Algorithms 3.3 and 3.4 describe proposed scheduler, MCS. It is hybrid of partitioned and global approaches. It partitions the periodic tasks amongst the processor cores statically using the well-known Bin Packing heuristic such as WFD. WFD is a widely used task allocation technique in literature where the focus is on dynamic energy reduction (Zapata and Alvarez, 2004). The reason behind the choice of WFD is that it balances the load uniformly on all the cores giving opportunity to scale the frequency and voltage. Aperiodic jobs follow global approach in which they are allocated to one of the cores dynamically and can migrate upon preemption if required. In this way, with the combined use of both the approaches, the utilization of each of the individual cores is increased and responsiveness of aperiodic jobs is improved.

The subset of periodic tasks after partitioning is denoted as S_i. Hence, S, the set of all periodic tasks in the system, is defined as $\{S_0, S_1, ...S_{M-1}\}$ where M is the number of cores. The algorithm maintains a set of M job queues, JQ = $\{JQ_0, JQ_1, ..., JQ_{M-1}\}$, one per core C_i. Each subset of tasks, S_i is assigned to its corresponding core C_i and all the jobs belonging to the tasks in S_i are inserted in job queue JQ_i.

Task scheduling involves making scheduling decisions on each core separately at scheduling points. The only scheduling decision points are completion of executing jobs or arrival of new jobs. The conditional statements at line number 5 and 7 in algorithm 3.4 shows the completion point and arrival point respectively. At each scheduling decision point, MCS computes dynamic core utilization U_{dyn_i} of processor core C_i using the formula in equations 3.11, 3.12 and 3.13 and selects the matching smallest frequency f_{opt} such that U_{dyn_i} is less than or equal to the scaling factor α where $\alpha = f_{opt}/f_{max}$, with $f_{opt} \in \{f_1, f_2, ..f_{max} \mid f_1 < f_2 < ... < f_{max}\}$. The discrete frequencies are denoted by f_i where i = 1 to the number of discrete frequencies in increasing order. The algorithm 3.3 shows frequency selection. The proposed method of dynamic core utilization calculation at any scheduling point gives the exact available utilization at that scheduling point and results in assignment of accurate frequency level to the scheduled task. Further if the next job to run on a particular core is a periodic job and if the aperiodic job queue is empty, then the core runs at the selected optimal frequency, else it runs at maximum frequency (refer line number 20-25). The dynamic utilization U_{dyn_i} for processor core C_i is calculated as:

$$U_{dyn\_i} = U_{future} + U_{past} \tag{3.11}$$

where,

$$U_{future} = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t,hp] \wedge\, !preempted\ J_i)} \frac{c_i}{hp-t} \tag{3.12}$$

$$U_{past} = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t,hp] \wedge preempted\ J_i)} \frac{c_{i\_rem}}{hp-t} \tag{3.13}$$

Where U_{past} denotes the sum of the remaining utilization of the periodic jobs which were released before time t and U_{future} denotes the sum of share of processor utilization of the periodic jobs that will be released during the interval [t, D_{PJ}). The notation J_i is the periodic job, c_{i_rem} is the remaining *wcet* of the preempted job, *hp* is hyper period and *t* is the current time.

Upon arrival of an aperiodic task, its virtual deadline is calculated using TBS algorithm, based on the current utilization of each core. It is denoted as v_{k_i} for aperiodic job A_k for core C_i and is defined by

$$v_{k\_i} = Max\{a_k, v_{k-1\_i}\} + \frac{c_k}{U_{srv\_i}} \tag{3.14}$$

Where $U_{srv\_i} = 1 - U_{dyn\_i}$, is the utilization of TBS on core C_i at the arrival time of aperiodic job A_k. This aperiodic job is then assigned to the processor core on which it finds earliest virtual deadline amongst all the cores as shown in equation 3.15.

$$v_{min\ \_index} = Min_{0 \le i \le M-1}\{v_{k\_i}\} \tag{3.15}$$

Where *min_index,* is the index of processor core that results in minimum virtual deadline among all the cores. Line numbers 10-17 in algorithm 3.4 handle calculation of virtual deadline on each core and assignment of aperiodic job to a processor core. Virtual deadline calculations and selection of core are done centrally by the aperiodic controller and requires very small overhead. This aperiodic controller has the knowledge of current utilizations of all the cores.

MCS is capable of scheduling the task set on a processor with single core. In this case, the periodic and aperiodic tasks are scheduled on a single core. The partitioning of periodic tasks is not required. Further, the aperiodic task assignment does not require finding a core that can provide minimum virtual deadline and does not have a situation in which migration of aperiodic job is required. We name the algorithm for uni-core processor as EE-UCS - Energy Efficient - Uni-Core Scheduler.

Algorithm 3.3: Frequency Selection of Proposed MCS Algorithm

```
Select_Frequency
Begin
1:    Calculate $U_{dyn\_i}$ from Eq. (3.11, 3.12, 3.13)
2:    select smallest freq $f_{opt} \in \{f_1, f_2, .. f_{max} \mid f_1 < f_2 < ... < f_{max}\}$
      such that $U_{dyn\_i} <= f_{opt}/f_{max}$
3:    Returns optimal frequency $f_{opt}$
End
```

Algorithm 3.4: Proposed Multi-Core Scheduling Algorithm (MCS)

Pre-Condition: S_i, set of periodic tasks belonging to core C_i after partition.

Post-Condition: Feasible schedule if exists; FAIL otherwise.

```
MCS
Begin
1:  Join all pending jobs corresponding to core $C_i$ in $JQ_i$
2:  if  Empty ($JQ_i$) then return IDLE $C_i$  end if
5:  if Curr.$JQ_i$.RemainingTime == 0 then
6:        Curr.$JQ_i$ = Head.$JQ_i$
7:  else if  Head.$JQ_i$.Priority > Curr.$JQ_i$.Priority then
8:        if  Curr.$JQ_i$ is periodic then
9:              preempt and insert Curr.$JQ_i$ in $JQ_i$
10:       else
11:             find core with earliest virtual deadline using eq. (3.14) and (3.15)
12:             if  $min\_index$ != i  then
13:                   preempt and migrate Curr.$JQ_i$ to core $C_{min\_index}$
14:             else
15:                   preempt and insert Curr.$JQ_i$ in $JQ_i$
16:             end if
17:       end if
18:       Curr.JQi = Head.JQi
19: end if
20: if ! DVFS  OR JQi has at least one aperiodic job OR Curr.$JQ_i$ is aperiodic job then
21:       execute Curr.JQi till Min(Curr.$JQ_i$.Remaining Time, Arrival of next job) at $f_{max}$
22: else
23:       $f_{opt}$ = Select_Frequency($C_i$)
24:       execute Curr.JQi till Min(Curr.JQi.Remaining Time / $\alpha$, Arrival of next job) at $f_{opt}$
25: end if
End
```

3.6.2 Schedule of Sample Task Set using MCS Algorithm

Assume a task set consisting of 3 periodic tasks (T_0, T_1, T_2) and 2 aperiodic tasks (A_0, A_1) on a dual core processor. The task attributes are shown in Table 3.1. Each task in the task set is described by its arrival time, period, *wcet*, deadline, processor core number to which it is assigned after partitioning, worst case utilization ($U_{wc} = c_i/p_i$), number of jobs in a hyper period and corresponding *aet* of each job. Hyper period of this task set is 50. The discrete frequencies assumed in this example are 40%, 50%, 70%, 90% and 100% of maximum frequency. Scheduler decisions at various scheduling decision points and the schedule are shown in table 3.2 and table 3.3 for Core 0 and Core 1 respectively.

According to table 3.2 and 3.3, periodic task T_1 is allocated to Core 0 while T_0 and T_2 are allocated to Core 1 using WFD algorithm. At time 0, job J_{10} arrives on Core 0 while J_{00} and J_{20} arrive on Core 1. At time 0, dynamic utilization of Core 0 and Core 1 are calculated as 0.6 using equation 3.11, so the optimal frequency of both cores is set to 70%. The *aet* of highest priority jobs on both cores (J_{10} on Core 0 and J_{20} on Core1) are scaled down to 70% frequency and are started on respective cores. At time 8, when aperiodic job A_0 arrives, virtual deadline is calculated for job A_0 on Core 0 and Core 1. The aperiodic job A_0 is allocated to Core 1 as it provides early virtual deadline than Core 0. Similarly, at time 20, due to arrival of higher priority periodic job J_{22}, A_0 is migrated to Core 0 as it finds an early virtual deadline on that core. In this way, the entire schedule is generated by following proposed MCS algorithm for dual core processor platform.

Table 3.1: Mixed Task Set (MCS)

Tasks	**Arrival**	**Period**	***wcet***	**Deadline**	**Core # (WFD)**	U_{wc}	**# of jobs in a HP**	***aet* of jobs**
T_0	0	25	10	25	Core 1	0.4	2	3, 7
T_1	0	50	30	50	Core 0	0.6	1	23
T_2	0	10	2	10	Core 1	0.2	5	1,1.2, 1.4, 1.6, 1.8
A_0	8	-	15	-		-	-	15
A_1	25	-	5	-		-	-	5

Table 3.2: Schedule on Core 0 using Proposed MCS Algorithm

Time	R/M	S/RP/C	Comp	Preempt (RAET)	UT(R/C)	Freq (%)	WCET, AET/ RAET	Scaled AET	VD
0	J_{10}/-	J_{10}/-/-	-	-	0.6/-	70	30, 23/-	32.86	-
8.0	A_0/-	-/-/J_{10}	-	-	0.58/-	70	-, -/17.4	24.86	43.79 (A_0 not Allocated)
10.0	-/-	-/-/J_{10}	-	-	-/-	70	-, -/16	22.86	40.59(A_0 not Allocated)
20.0	-/A_0	A_0/-/-	-	J_{10}(9.03)	0.53/-	100	-, -/4.2	4.2	29 (A_0 Migrated from Core 1to Core 0)
24.2	-/-	-/J_{10}/-	A_0	-	-/0.62	70	-, -/9.03	12.9	-
25.0	A_1/-	A_1/-/-	-	J_{10}(8.47)	-/-	100	5, 5/-	5	42 (A_1 Allocated)
30.0	-/-	-/ J_{10}/-	A_1	-	-/0.77	90	-, -/8.47	9.41	-
39.4	-/-	-/-/-	J_{10}	-	-/-	-	-, -/-	-	-
CPU is idle from 39.4 to 50									

R/M - Release/Migration, S/RP/C - Started/Resumed after Preemption/Continued, Comp - Completed, Preempt (RAET) - Preempted (Remaining Actual Execution Time at maximum frequency), UT(R/C) - Dynamic Core Utilization (upon Release/upon Completion), Freq (%) - Normalized Frequency in %, WCET, AET/RAET - Worst Case Execution Time, Actual Execution Time/Remaining Actual Execution Time at maximum frequency, Scaled AET - Actual Execution Time at scaled frequency, VD - Virtual Deadline of aperiodic task.

Table 3.3: Schedule on Core 1 using Proposed MCS Algorithm

Time	R/M	S/RP/C	Comp	Preempt (RAET)	UT(R/C)	Freq (%)	WCET/ AET/ RAET	Scaled AET	VD
0	J_{00},J_{20}/-	J_{20}/-/-	-	-	0.6/-	70	2/1/-	1.4	-
1.4	-/-	J_{00}/-/-	J_{20}	-	-/0.58	70	10/3/-	4.3	-
5.7	-/-	-/-/-	J_{00}	-	-/0.41	-	-/-/-	-	(CPU Idle)
8.0	A_0/-	A_0/-/-	-	-	0.43/-	100	15/15/-	15	34.25 (A_0 Allocated)
10	J_{21}/-	J_{21}/-/-	-	A_0 (13)	-/-	100	2/1.2/-	1.2	-
11.2	-/-	-/A_0/-	J_{21}	-	-/-	100	-/-/13	13	34.25
20.0	J_{22}/-	J_{22}/-/-	-	A_0 (4.2)	0.53/-	70	2/1.4/-	2	34.25 (A_0 Migrated to Core 0)
22.0	-/-	-/-/-	J_{22}	-	-/-	-	-/-/-	-	- (CPU Idle)
25.0	A_1,J_{01}/-	J_{01}/-/-	-	-	0.56/-	70	10/7/-	10	45.6 (A_1 not Allocated)
30	J_{23}/-	J_{23}/-/-	-	J_{01} (3.5)	0.52/-	70	2/1.6/-	2.3	-
32.3	-/-	-/J_{01}/-	J_{23}	-	-/0.48	50	-/-/3.5	7	-
39.3	-/-	-/-/-	J_{01}	-	-/-	-	-/-/-		(CPU Idle)
40.0	J_{24}/-	J_{24}/-/-	-	-	0.2/-	25	2/1.8/-	7.2	-
47.2	-/-	-/-/-	J_{24}	-	-	-	-	-	-
CPU is Idle from 47.2 to 50									

R/M - Release/Migration, S/RP/C - Started/Resumed after Preemption/Continued, Comp - Completed, Preempt (RAET) - Preempted (Remaining Actual Execution Time at maximum frequency), UT(R/C) - Dynamic Core Utilization (upon Release/upon Completion), Freq (%) - Normalized Frequency in %, WCET, AET/RAET - Worst Case Execution Time, Actual Execution Time/Remaining Actual Execution Time at maximum frequency, Scaled AET - Actual Execution Time at scaled frequency, VD - Virtual Deadline of aperiodic task.

3.6.3 Correctness Proof and Schedulability of MCS Algorithm

Theorem 1: A system of independent preemptable mixed tasks containing N periodic and M aperiodic tasks, where periodic tasks have implicit deadlines and are in-phase and aperiodic tasks have soft deadlines and arrives arbitrarily, is schedulable on K cores according to MCS if the periodic tasks are partitionable among K processor cores using Bin Packing heuristics, and the total utilization of periodic tasks and aperiodic tasks is less than K.

Proof: MCS follows WFD partitioning heuristics which guarantee the assignment of tasks among K cores. Since the Bin Packing partitioning heuristics provide the utilization bound of 66% (Andersson and Tovar, 2006), the remaining utilization of each processor core is used for executing aperiodic tasks thereby increasing the schedulability of the task set.

MCS follows EDF scheduling policy for scheduling periodic tasks on each processor core independently. On the other hand, TBS scheduling policy is used for scheduling aperiodic tasks along with periodic tasks where the utilization of TBS server is equal to the remaining utilization of the processor core other than the assigned utilization of periodic tasks.

As EDF along with TBS scheduling policies are optimal (Liu, 2008) for uniprocessors, bin packing heuristic allocates periodic tasks on multiple cores with utilization bound of 66% and aperiodic tasks use the remaining utilization of processor cores, MCS produces a feasible schedule on multi-core processor and increases the utilization bound of the processor cores.

Theorem 2: In a system of independent preemptable mixed tasks consisting of periodic and aperiodic tasks where periodic jobs follow EDF schedule and aperiodic jobs follow TBS schedule, the dynamic energy optimization guarantees least idle time and is achieved by DVFS.

Proof: The MCS algorithm selects the optimal frequency and voltage at each scheduling point on the basis of dynamic utilization of the processor core at that scheduling point. The dynamic utilization, frequency and voltage are calculated on the basis of current load of the processor core which results in minimum idle intervals. The correctness of dynamic utilization of a processor core is proved by using mathematical induction in corollary 1.

Corollary 1: When a task set *T* contains *N* periodic tasks and *M* aperiodic tasks where $N \geq 2$ and $M > 0$, the dynamic utilization of the processer at any scheduling time t is given by (from equations 3.11, 3.12 and 3.13)

$$U_{dyn\_i} = U_{future} + U_{past}$$

Where

$$U_{future} = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t,hp] \wedge \, !preempted \; J_i)} \frac{c_i}{hp-t}$$

$$U_{past} = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t,hp] \wedge \, preempted \; J_i)} \frac{c_{i\_rem}}{hp-t}$$

Proof:

Assume that task set *T* contains (*N*+*M*) tasks where $N \geq 2$ and $M > 0$. The dynamic utilization stated in the above equations is proved by using mathematical induction as follows.

The mathematical induction is based on the number of periodic jobs to be executed during the interval [*t,hp*) where *t* is any scheduling point.

Basic Step: At any scheduling time *t*, if there are no lower priority jobs waiting in the job queue other than scheduled periodic job *PJ* and there is no job that is to be executed after the scheduled job during interval [*t,hp*) then

$$U_{past} = \frac{c_{PJ}}{hp-t} \qquad (3.16)$$

$$U_{future} = 0 \qquad (3.17)$$

Therefore, dynamic utilization of the core C_i is equal to the ratio of *wcet* of the scheduled periodic job *PJ* and remaining time from scheduling point *t* to *hp*.

Inductive Step: At any scheduling time *t*, Let *j* be the number of lower priority jobs waiting in the job queue including scheduled job and k be the number of jobs arriving during interval [*t,hp*). In this case, dynamic utilization can be stated as follows:

$$U_{past} = \sum_{i=1}^{j} \frac{c_{i\_rem}}{hp-t} \qquad (3.18)$$

$$U_{future} = \sum_{i=1}^{k} \frac{ci}{hp-t} \qquad (3.19)$$

Therefore, dynamic utilization at any scheduling time t will be the sum of U_{past} and U_{future}.

Theorem 3: In a multi-core system where independent preemptable periodic tasks are assigned and scheduled independently on different cores using MCS with DVFS and aperiodic jobs are globally scheduled, the response time of aperiodic jobs are not affected due to voltage and frequency scaling.

Proof: MCS schedules aperiodic jobs globally to any of the cores on multi-core system on the basis of virtual deadline that aperiodic job receives on each core. It assigns the job to a core which gives early virtual deadline. The correctness of virtual deadline computation for an aperiodic job is given in corollary 2. All aperiodic jobs execute at maximum voltage and frequency which results in minimum response time.

Corollary 2: In a system consisting of *K* processor cores, *N* periodic tasks and *M* aperiodic tasks, the virtual deadline for an aperiodic job is computed as follows:

$$v_{k\_i} = \mathrm{Max}\{a_k, v_{k-1\_i}\} + \frac{c_k}{U_{srv\_i}}$$

Proof: The virtual deadline of any aperiodic job depends on the TBS utilization. For example, in a dual core system, if server utilization of core 1 is greater than server utilization of core 2, then core 1 provides early virtual deadline than core 2. From the above equation, it is observed that server utilization depends on core's dynamic utilization; the correctness of dynamic utilization calculation can be proved by corollary 1.

3.6.4 Algorithmic Complexity of MCS Algorithm

The algorithmic complexity of the proposed MCS is the product of the number of scheduling decision points and the complexity per scheduling decision. Given a task set S with hyper period H, the algorithmic complexity of MCS is

$$T_{MCS}(S,H) = Decisions\ (MCS,S,H) \times\ C_{decision}\ (MCS,S,H) \quad (3.20)$$

$$Decisions\ (MCS,S,H) = N(A_p +\ A_a) +\ NC + NP \quad (3.21)$$

Where *Decisions* is the number of decision points to be made by the algorithm and $C_{decision}$ is the time complexity of a single decision. *Decisions* can be divided into four types of decisions points: number of periodic job arrivals *N(A_p)* and number of aperiodic job arrivals *$N(A_a)$*, number of completion points (*NC*) and number of aperiodic preemption points (*NP*). Time complexity of MCS can be derived as follows:

$$T_{MCS}(S,H) = O\left(N\left(A_p + A_a\right) \times (T \log_2 T)\right) + O\left(NC \times (K+P)\right) + O(NP \times T \log_2 T) + O(L) + O(M) \quad (3.22)$$

where T is the number of tasks in a task set S. The time required for insertion in job queues upon job arrival is O (T $\log_2$T). The constant K is the number of jobs appearing in a time interval [*t, hyper period*) at any scheduling time t and P is the number of lower priority jobs waiting in the job queue at time *t*. The number of time units required to calculate dynamic utilization at any completion time are (K + P). In practice, the value of K and P are very small as compared to total jobs. On preemption, the aperiodic jobs are inserted in the job queue of one of the cores. This additional insertion time is O (NP * T$\log_2$T). The time complexity of the selection of matching frequency level from the L discrete frequency levels is O (L). O (M) is the constant amount of time required to select a processor core which provides earlier virtual deadline to a scheduled aperiodic job where M is the number of processor cores.

3.7 Parameters for Comparison

This section presents the details of the parameters used to validate the performance of the proposed algorithms, EEDVFS, EE-UCS and MCS over other existing algorithms like non-DVFS, Static Voltage and Frequency Scaling (SVFS) and cycle conserving EDF with TBS scheduling policy for multi-core processors. The non-DVFS multi-core scheduler executes the jobs at maximum frequency, SVFS executes the jobs at a frequency selected based on worst case utilization of the task set, cycle conserving EDF executes the jobs at a frequency based on the utilization formula discussed in Pillai and Shin (2001). These parameters include aperiodic response time, number of scheduling decision points, number of preemption points, number of migration points and energy consumption.

3.7.1 Response Time

The response time of a job is an important performance metric for any real time application. Specifically, it is a critical measure for aperiodic jobs as they have soft deadline. In order to achieve good overall performance of a real time application, response time of aperiodic jobs must be taken care of along with the hard deadlines of periodic jobs. In this work, the proposed algorithms ensure early response time of all aperiodic jobs along with meeting the hard deadlines of periodic jobs.

The response time of any job can be defined as

$$\textit{Response Time } (J_i) = \textit{Finish Time } (J_i) - \textit{Arrival Time } (J_i) \qquad (3.23)$$

where J_i is any periodic or aperiodic job in a task set. We have focused on response time of aperiodic jobs. For the purpose of evaluation, the maximum aperiodic response time per task set is defined as -

$$\textit{Maximum Aperiodic Response Time } (T) = \max(\textit{Response Time } (AJ_i)) \qquad (3.24)$$

where T is a mixed task set, AJ_i is an aperiodic job in the task set T. For example, let a task set T contains 3 periodic tasks and 2 aperiodic jobs, the response times of the two aperiodic jobs be $AJ_1 = 5$ and $AJ_2 = 3$. The maximum aperiodic response time of task set T should be 5. The maximum response time is normalized with respect to execution time of the corresponding aperiodic job. The normalized response time value of 1 corresponds to the least possible response time.

3.7.2 Scheduling Decision Points

The number of scheduling decisions made during entire schedule of a task is an important parameter which is required to validate the performance of the scheduler. At each scheduling decision point, scheduler decides which job to execute; this decision takes additional time and energy. For example, if a job is preempted and requires migration to other core, this decision and action takes time and energy. This is called as scheduler overhead. As our focus is on saving energy consumption, we measure energy consumed by the scheduler while taking scheduling decisions. The energy consumed by various scheduling events such as periodic job arrivals, aperiodic job arrivals, job completions, preemptions with cold cache job, preemptions with hot cache job and job migrations are considered while performing the simulations. The energy consumed by each scheduling event is explained in more detail in section 3.7.5.

3.7.3 Preemption Count

Preemptive scheduling is one of the key aspects of any real time scheduling algorithm. Preemptive scheduling algorithms perform better than the non-preemptive algorithms on time-sharing real time systems where multiple real time tasks are running simultaneously on one or more processor cores. For example, EDF, an optimal real time scheduling algorithm for uniprocessor platform is a preemptive algorithm. On the other hand, preemptive scheduling incurs significant overhead caused due to context switching. This includes the time taken for suspending the running job and dispatching a new one and cache misses which includes the time taken to transfer the requested page from lower level cache to the higher-level cache. We have considered the energy consumed by the preemption overhead while performing the simulations. The calculation for energy consumed due to preemption is explained in detail in section 3.7.5.

3.7.4 Migration Count

In global multiprocessor/multi-core scheduling, the jobs are permitted to migrate from one processor core to another. The advantage of global scheduling is that it increases overall system utilization but at the same time migration of tasks incurs overhead due to information flow between the processor cores. The overhead is in terms of time and energy. The use of shared memory architecture would incur considerably less overhead than distributed memory architecture. Migration of a job requires transfer of data/instruction from one processor core to another. In shared memory architecture where L2 cache is shared among all the processor cores, transfer of data/instruction would happen from shared L2 cache to private L1 cache of another target processor core. But, in non-uniform memory architecture, the latency of transfer of data/instruction varies between the cores and would typically take longer time than the shared memory architecture. The simulations carried out in this work assume shared memory architecture and includes the overheads caused due to migration of jobs. The energy consumed by the migration overhead is explained in detail in section 3.7.5.

3.7.5 Energy Consumption

The overall dynamic energy consumption includes energy consumed during task execution and other scheduling events such as arrival, completion, preemption, migration and scheduling decision. Energy consumption during task execution, E_{task_exe}, is calculated in various intervals over the hyper period at different frequencies and voltages. For calculating energy consumption of the scheduler, the number of task arrivals, completions, preemptions, migrations and scheduling decision points are computed throughout the hyper period for each task set. Equation 3.25 shows the formula for calculating total dynamic energy consumption.

$$Dynamic\ Energy\ Consumption = E_{task\_exe} + T_{sched} \times E \tag{3.25}$$

where, T_{sched} is total time spent in scheduling events, E is the energy consumed per unit time by the processor running at maximum frequency. The total time spent on scheduling events is calculated as follows:

$$T_{sched} = \mathrm{K1} \times \mathrm{p_{ac}} + \mathrm{K2} \times \mathrm{ap_{ac}} + \mathrm{L} \times \mathrm{cc} + \mathrm{M1} \times \mathrm{pc_{cold}} + \mathrm{M2} \times \mathrm{mc} + \mathrm{P} \times \mathrm{dc} + \mathrm{CST} \times (\mathrm{cc} + \mathrm{pc\_cold} + \mathrm{pc\_hot} + \mathrm{mc}) \tag{3.26}$$

where, p_{ac}, ap_{ac}, cc, pc_{cold}, mc, dc and pc_{hot} are the number of periodic job arrival points, aperiodic job arrival points, completion points, preemption points with cold cache job, migration points, decision points and preemption points with hot cache job respectively. The time constants CST, K1, K2, L, M1, M2 and P are time required for context switching, periodic job arrival, aperiodic job arrival, completion of a job, transfer data/instruction from L2 cache to L1 cache upon preemption, transfer data/instruction from L2 cache to L1 cache upon migration and the scheduler to make a scheduling decision respectively. These time constants and execution times are measured in milliseconds. The time constants are calculated by first running the scheduler code for large number of iterations and then taking average of all the iterations. These time constants are shown in Table 3.4. Preemption and migration cost of a job is considered as ten times the CST. It is assumed that the preemption and migration demand transfer of at least 3 and 5 pages respectively.

The simulations are conducted by considering the frequency and voltage values of Transmeta Crusoe processor with maximum frequency of 3.1 GHz and maximum supply voltage of 1 V. The critical speed of Transmeta Crusoe processor is 1.26 GHz which is 41% of maximum speed. The voltage corresponding to critical speed is 0.7 V (Jejurikar et al., 2004). MCS takes care of critical speed and does not slow down the processor below this speed. The details of frequency and voltage ranges are given in Table 3.5. Table 3.6 shows the energy consumption of the example task set (Table 3.1) with MCS, Cycle-Conserving (Pillai and Shin, 2001), Non-DVFS and Static-VFS algorithms over one hyper period.

Table 3.4: Scheduler Time Constants

Time Constants	Value (in ms)
K1	0.002012
K2	0.012074
L	0.000344
M1	0.280506
M2	0.4675107
P	0.0010041
CST	0.0093502

Table 3.5: Frequency/Voltage Settings of 70nm Transmeta Crusoe Processor

Level	Frequency (GHz)	Voltage (V)
0	3.1	1.0
1	2.79	0.95
2	2.48	0.9
3	2.17	0.85
4	1.86	0.8
5	1.5	0.75
6	1.26	0.7

Table 3.6: Energy Consumption over a Hyper Period (MCS)

Algorithm	Execution Energy (m Joules)	Scheduler Energy (m Joules)	Total Energy (m Joules)
Proposed MCS	**69.94**	**3.49**	**85.938**
Non-DVFS	79.98	3.21	98.965
Cycle Conserving	72.53	3.21	89.696
SVFS	73.65	3.21	91.359

3.8 Experimental Setup

The synthetic task generator generates mixed task sets for experimentation. The task set generation algorithm first obtains uniformly distributed utilization values for each task in a task set (Emberson et al., 2010) and sums them to a total load of the task set. Period of each periodic task is chosen to be a random number such that it is a natural factor of a given hyper period value. Arrival time of the aperiodic task is obtained as a random number over a hyper period and the minimum inter-arrival time between two aperiodic tasks is not more than 5% to 10% of the hyper period. The execution time of each periodic task is derived from the utilization and period. The synthetic task generator generates periodic tasks for a wide range of utilizations: 30% to 80% and the aperiodic tasks utilization is based on the remaining utilization of processor. The number of periodic tasks in a task set range from 2 to 20 and aperiodic tasks in a task set range from 2 to 6. For each class of fixed number of tasks and utilization, 100 task sets are generated.

A real-time mixed task set simulator, STREAM, (Digalwar et al., 2015) is designed and implemented to run the synthetic benchmark programs using various schedulers. The schedulers used in the simulator for evaluation are MCS, Cycle-Conserving, Non-DVFS and SVFS. All the schedulers use EDF for scheduling periodic jobs and TBS for scheduling aperiodic jobs. The simulator uses a local queue per core for periodic jobs which is filled using WFD partitioning method. It uses a global queue for aperiodic jobs. In the cycle conserving algorithm, frequency is selected at each scheduling point based on the dynamic utilization method discussed in Pillai and Shin (2001). In non-DVFS scheduling technique, the jobs are always executed at maximum frequency and in static VFS scheduling technique, jobs are executed at a predefined frequency such that total worst case utilization of periodic tasks is less than or equal to scaling factor α where $\alpha = f_{static}/f_{max}$, $f_{static} \in \{f_1, f_2, .. f_{max} \mid f_1 < f_2 < ... < f_{max}\}$.

The simulator measures various parameters like number of job (periodic and aperiodic) arrivals, job departures, scheduling decisions, preemptions with cold cache, preemptions with hot cache and migrations. It also measures the overall energy consumption and response time of aperiodic jobs. These parameters are measured for MCS, Cycle-Conserving, non-DVFS and Static VFS schedulers on task sets generated by the synthetic task set generator.

3.9 Results and Discussion

This section analyzes the performance of the proposed scheduling algorithms, EEDVFS, EE-UCS and MCS. The algorithms, EEDVFS and EE-UCS are compared for energy consumption and response time. The algorithm, MCS is compared with Cycle-Conserving (Pillai and Shin, 2001), non-DVFS and SVFS (Pillai and Shin, 2001) schedulers based on the parameters of comparison explained in section 3.7. The energy is measured in milli-joules where as the response time is measured in milli-seconds. The average energy consumption and average response time values presented in the graphs are normalized with respect to hyper period and execution time respectively. The equation 3.27 and equation 3.28 show the formulation of Normalized Energy Consumption (*NEC*) of a task set and Normalized Response Time (*NRT*) of an aperiodic task with maximum response time in a task set. Let *n* be the number of aperiodic tasks in a task set and A_i represents *i*th aperiodic task in a task set.

$$NEC\ of\ a\ task\ set = \frac{Total\ Energy\ consumption\ of\ the\ task\ set}{Hyper\ period\ of\ the\ task\ set} \qquad (3.27)$$

$$NRT\ of\ aperiodic\ task\ in\ a\ task\ set = \frac{Max_{0<i\leq n-1}(Response\ Times\ (A_i)}{WCET\ of\ Max_{0<i\leq n-1}(Response\ Times\ (A_i)} \qquad (3.28)$$

3.9.1 Performance Analysis of Proposed Uniprocessor based Scheduling Algorithms

The graphs in figures 3.1 to 3.6 show the result of investigation of the energy efficient scheduling algorithms for uni-processor platform. The analysis shows the amount of energy consumption and response time of aperiodic tasks by increasing the periodic tasks, total periodic utilization and aperiodic tasks. From the figures, it can be observed that EE-UCS performs better than EEDVFS in terms of energy saving and response time. This is because, EE-UCS is able to reclaim the dynamic slack more accurately than EEDVFS and TBS used in EE-UCS utilizes the server budget more efficiently than DS in EEDVFS.

In figures 3.1 and 3.2, energy consumption and response time increase with increase in periodic tasks due to increasing scheduling overheads. In figures 3.3, the energy consumption increases with increasing periodic utilization as the execution energy increases with utilization. The response time of aperiodic tasks also increases with increase in utilization as shown in figure 3.4 due to increase in total execution time. From figures 3.5, it can be observed that there is little increase in energy consumption with increase in number of aperiodic tasks in both the algorithms (EE-UCS and EEDVFS) as the utilization of aperiodic tasks is fixed and scheduling overhead incurs small amount of energy. In case of response time in figure 3.6, sufficient increase is visible for EEDVFS where EE-UCS does not have more effect. This is due to the use of DS in EEDVFS which takes longer response time as compared to EE-UCS which uses TBS.

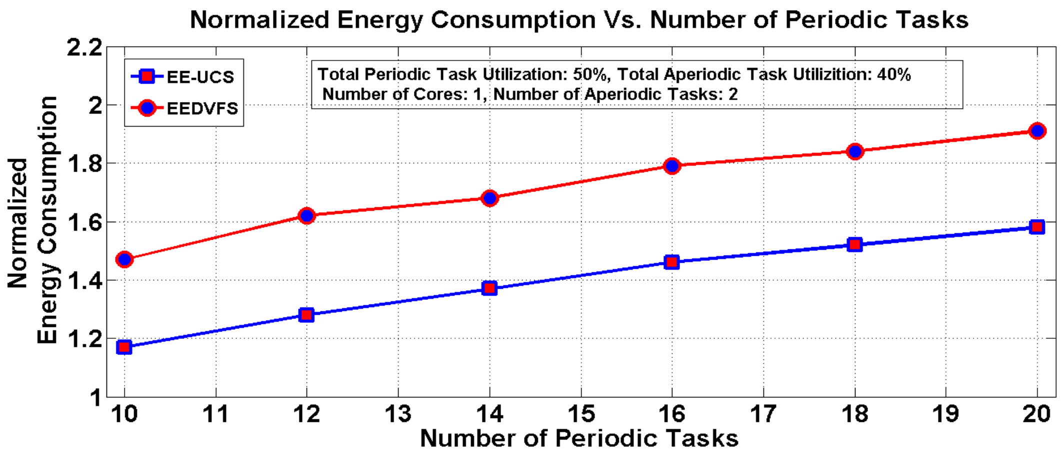


Figure 3.1: Normalized Energy Consumption Vs. Number of Periodic Tasks (Uni-Core)

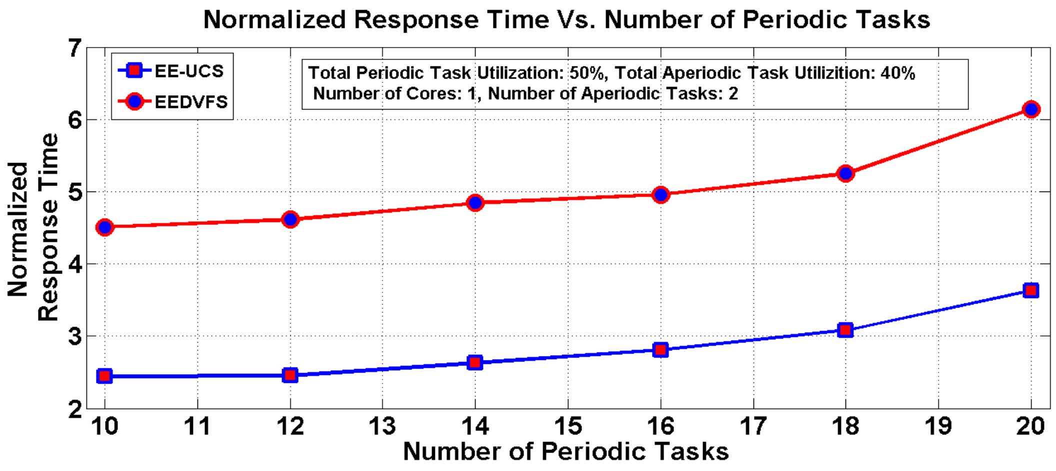


Figure 3.2: Normalized Response Time Vs. Number of Periodic Tasks (Uni-Core)

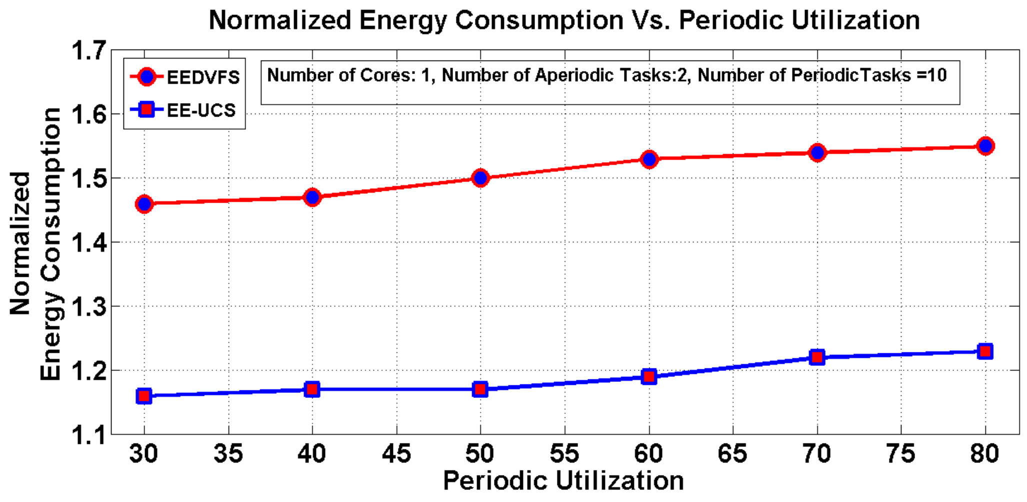


Figure 3.3: Normalized Energy Consumption Vs. Periodic Utilization (Uni-Core)

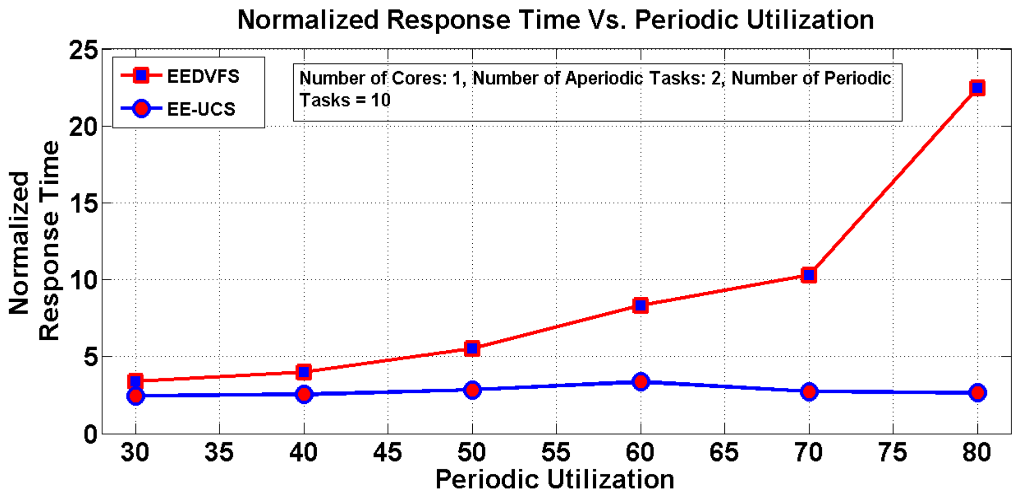


Figure 3.4: Normalized Response Time Vs. Periodic Utilization (Uni-Core)

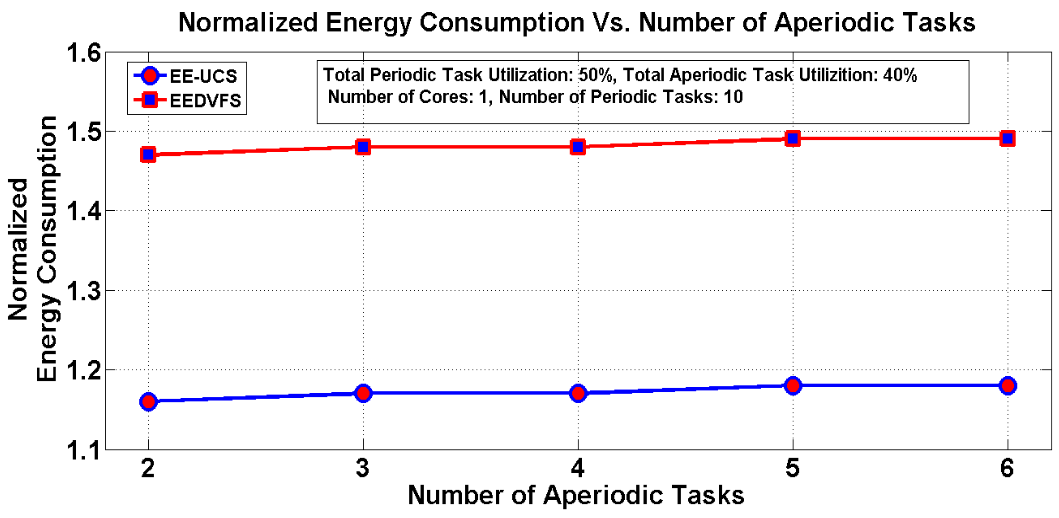


Figure 3.5: Normalized Energy Consumption Vs. Number of Aperiodic Tasks (Uni-Core)

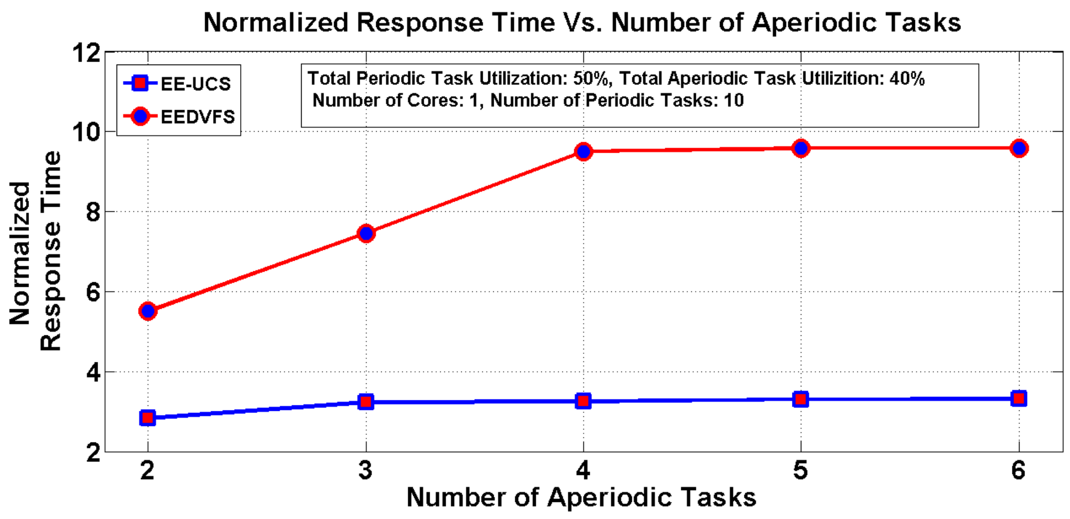


Figure 3.6: Normalized Response Time Vs. Number of Aperiodic Tasks (Uni-Core)

3.9.2 Performance Analysis of Proposed Multi-core Scheduling Algorithm (MCS)

The graphs shown in figures 3.7, 3.8, 3.9 and 3.10 show the effect on energy consumption by increasing number of periodic tasks in a task set, increasing periodic utilization, increasing number of aperiodic tasks and increasing number of processing cores.

3.9.2.1 Effect on Energy Consumption

The plot in figure 3.7 shows the comparison of energy consumption measured for MCS with three other scheduling algorithms. The energy is measured by increasing the periodic tasks in a task set while keeping the total utilization of periodic tasks fixed at 50% and keeping the remaining 50% utilization reserved for aperiodic tasks. The number of aperiodic tasks and processing cores are fixed to 2 and 8 respectively. Two observations can be made from the graph in figure 3.7. First, the proposed MCS algorithm results in more energy saving than other algorithms due to its efficient dynamic frequency selection mechanism. Second, irrespective of the algorithms, the energy consumption slightly increases with increasing number of periodic tasks because of the scheduling overhead. The number of preemption counts and scheduling points increase as number of periodic tasks are increases which consume additional energy.

In figure 3.8, the energy consumption is measured by increasing the utilization of periodic tasks in a task set by keeping number of periodic tasks, aperiodic tasks and the number of processing cores fixed to 16, 2 and 8 respectively. In each case, the remaining utilization is reserved for aperiodic server. The graph in figure 3.8 shows that the energy efficient algorithm MCS consumes less energy than the other algorithms. Irrespective of the choice of algorithms, the energy consumption increases with increasing periodic utilization. This is because, the execution energy increases with increase in total utilization.

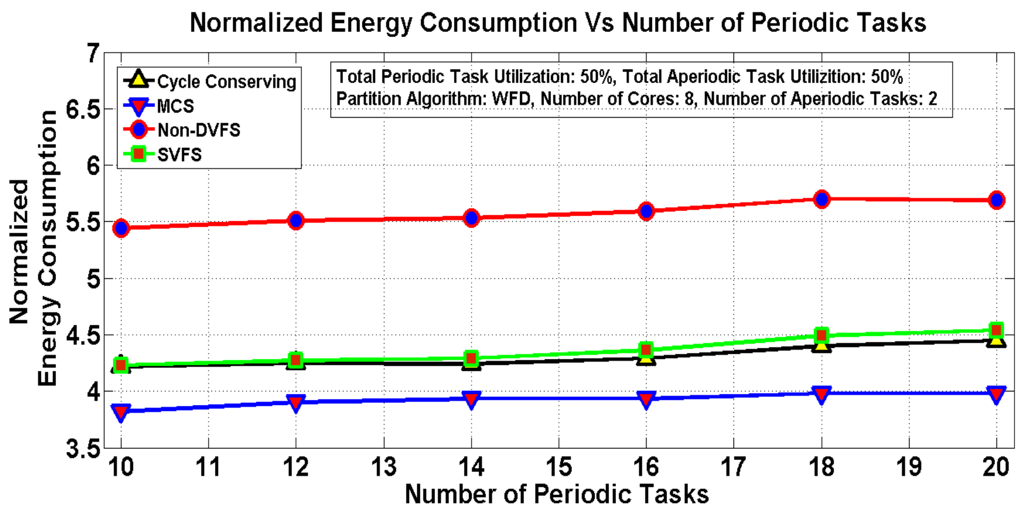


Figure 3.7: Normalized Energy Consumption Vs. Number of Periodic Tasks

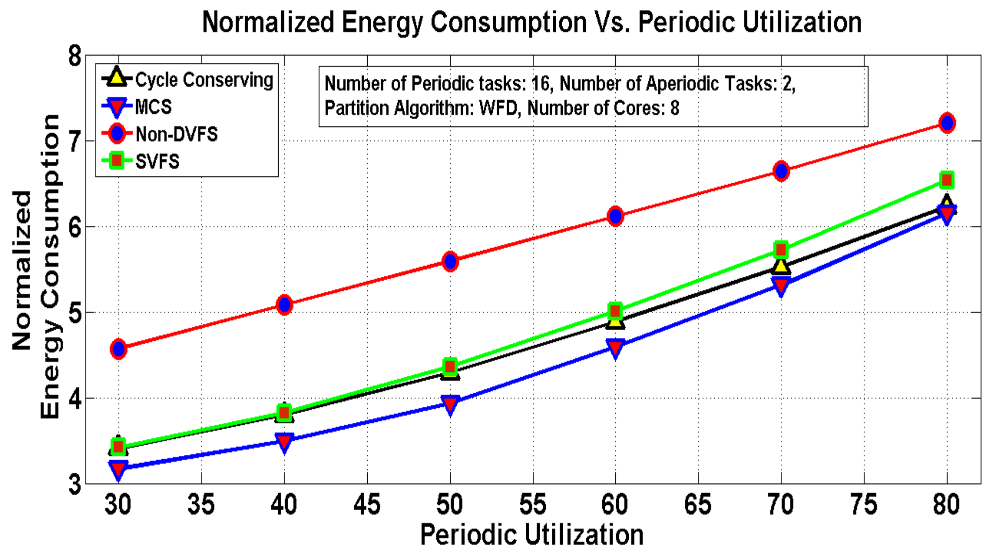


Figure 3.8: Normalized Energy Consumption Vs. Periodic Utilization

The graphs in figures 3.9 and 3.10 show energy consumption with respect to two parameters: increase in number of aperiodic tasks and increase in number of processing cores. In both the cases, the number of periodic tasks is fixed to 16 and total periodic utilization is fixed to 50%. The energy consumption measured for MCS is less as compared to other algorithms. There is no significant increase in energy consumption by increasing number of aperiodic tasks as this does not highly increase the scheduling events as well as scheduling overheads. The effect of increasing number of cores is very little on the overall energy consumption. This is because of increase in static energy consumed during idle state of processor cores. The idle state of processor cores increase with increase in number of cores.

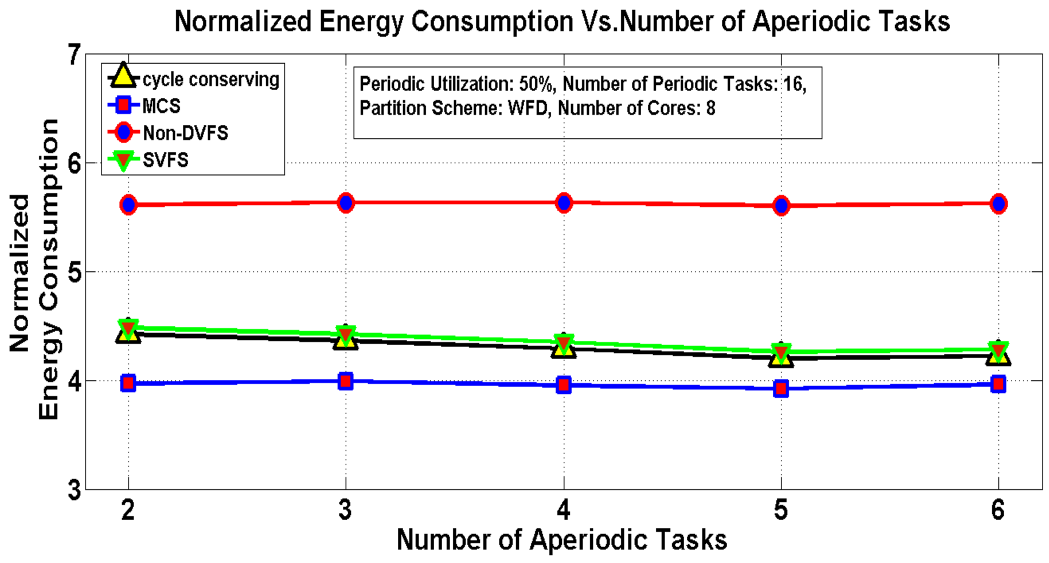


Figure 3.9: Normalized Energy Consumption Vs. Number of Aperiodic Tasks

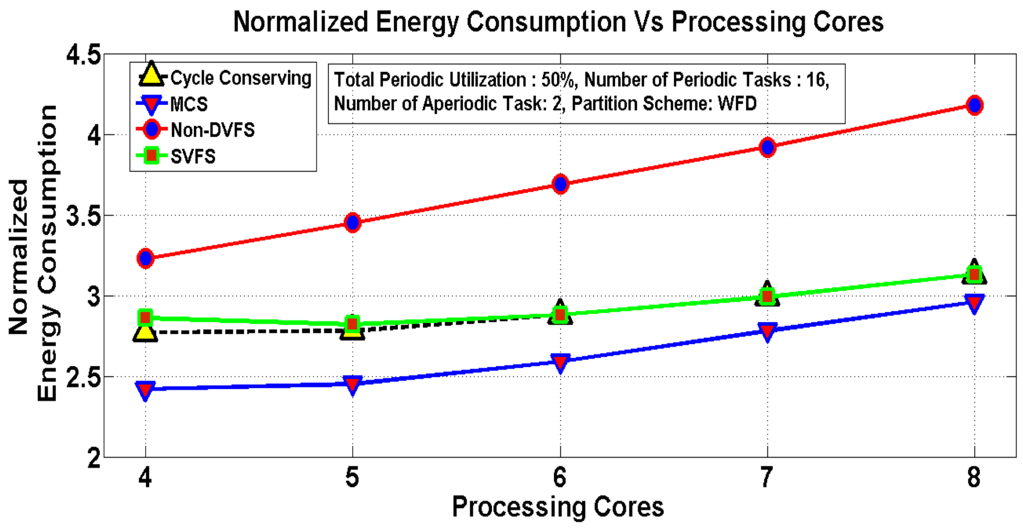


Figure 3.10: Normalized Energy Consumption Vs. Processing Cores

3.9.2.2 Effect on Response Time

Figure 3.11 shows the effect of increasing number of periodic tasks in a task set on normalized response time of aperiodic tasks. By increasing the number of periodic tasks, there exist a rise in scheduling events such task arrivals, completions, preemptions etc., which results in longer response time. Therefore, for all the algorithms, the response time increases with increase in number of periodic tasks. MCS gives nearly equal response time as compared to other energy aware and non-energy aware algorithms. This is because two reasons. One is, MCS executes periodic job at maximum speed in presence of aperiodic job and aperiodic job is always executed at maximum speed. Only in absence of aperiodic jobs, periodic jobs are executed at a scaled frequency. The other reason of nearly equal response time is because of efficient calculation of dynamic utilization of processor cores which results in accurate selection of core for aperiodic task execution.

The graph in figure 3.12 shows the effect of increasing periodic utilization on the response time of aperiodic tasks. It keeps the number of periodic and aperiodic tasks fixed at 16 and 2 respectively. It can be observed from the graph that the response time increases with increase in periodic utilization irrespective of the algorithms. This is due to increase in execution time of the periodic tasks in a task set which in turn increases the response time. The normalized response time achieved by all the algorithms is nearly equal and it is ranging between 1.1 and 2.1. That is, in case of lower utilization, it is nearly equal to optimal response time value of 1.

Figure 3.13 shows normalized response time of aperiodic tasks vs. increasing number of processing cores. It can be observed from the graph in figure 3.13 that it takes longer response time for less number of cores and as the cores increase, the response time decreases. This happens because the allocation technique allocates the aperiodic tasks to lightly loaded core. With WFD partitioning scheme, as the number of cores is increased for a task set of fixed utilization, the possibility of finding lightly loaded core increases and thus response time decreases. Also, as MCS uses processor's current utilization, it finds the lightly loaded core more accurately than Non-DVFS and SVFS algorithms. As a result, in most of the cases, MCS is having shorter response time than the other algorithms.

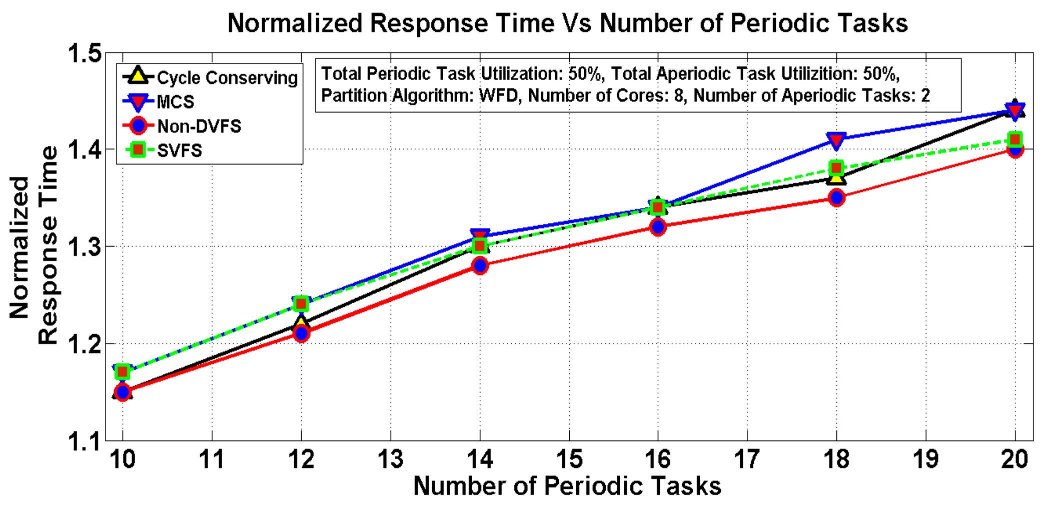


Figure 3.11: Normalized Response Time Vs. Number of Periodic Tasks

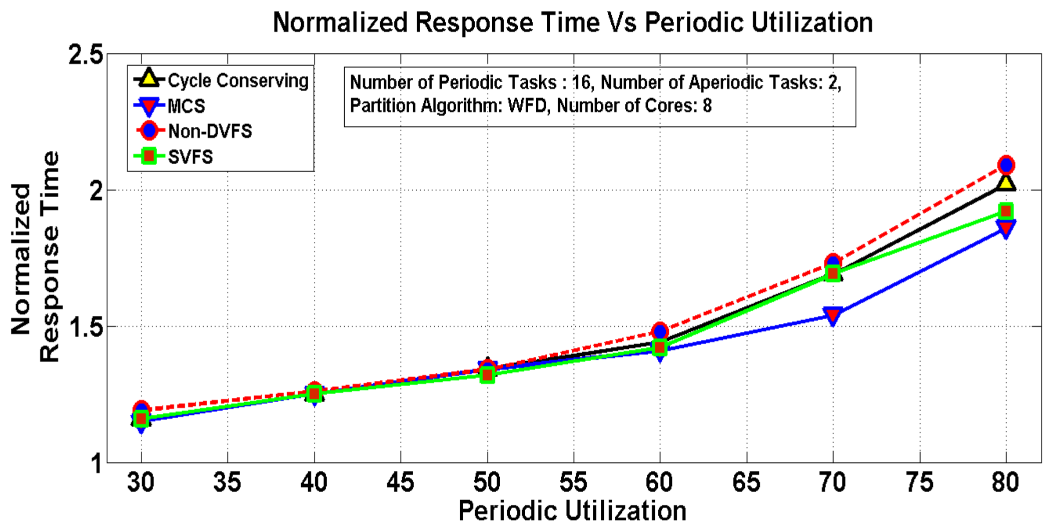


Figure 3.12: Normalized Response Time Vs. Periodic Utilization

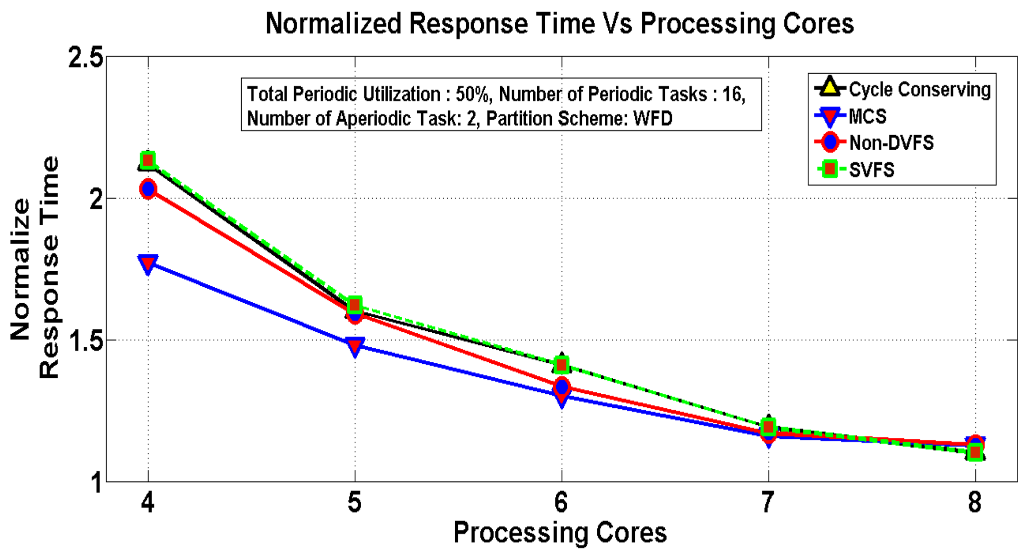


Figure 3.13: Normalized Response Time Vs. Processing Cores

3.9.2.3 Effect on Scheduling Events

The analysis of scheduling overhead in terms of number of preemptions, migrations and scheduling decisions is performed and the results are shown in the following graphs. The graphs in figures 3.14 and 3.15 show the variations in number of preemptions due to increase in number of periodic tasks and increase in periodic utilization. In both the cases, the preemption count increases with increase in periodic tasks and utilization. This is because, by increasing the number of tasks in a task set, the probability of arrival of higher priority jobs increases resulting in increase in number of preemptions. With increase in periodic utilization, the total execution time increases which results in larger preemption count.

In figure 3.16, the number of scheduling decisions is shown against increasing number of periodic tasks. As number of periodic tasks increase, the number of jobs per task over the hyper period also increases thereby increasing number of arrival and completion events. As a result, scheduling decision points increase.

Among all, MCS has more number of preemptions and scheduling decision points because it takes more execution time as compared to other algorithms due to frequency scaling.

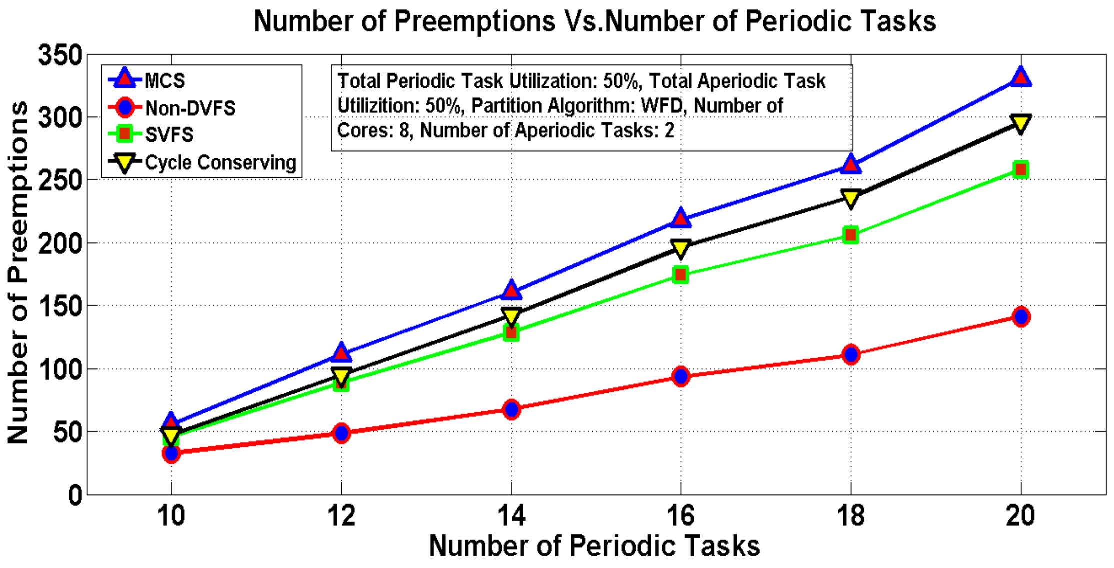


Figure 3.14: Number of Preemptions Vs. Number of Periodic Tasks

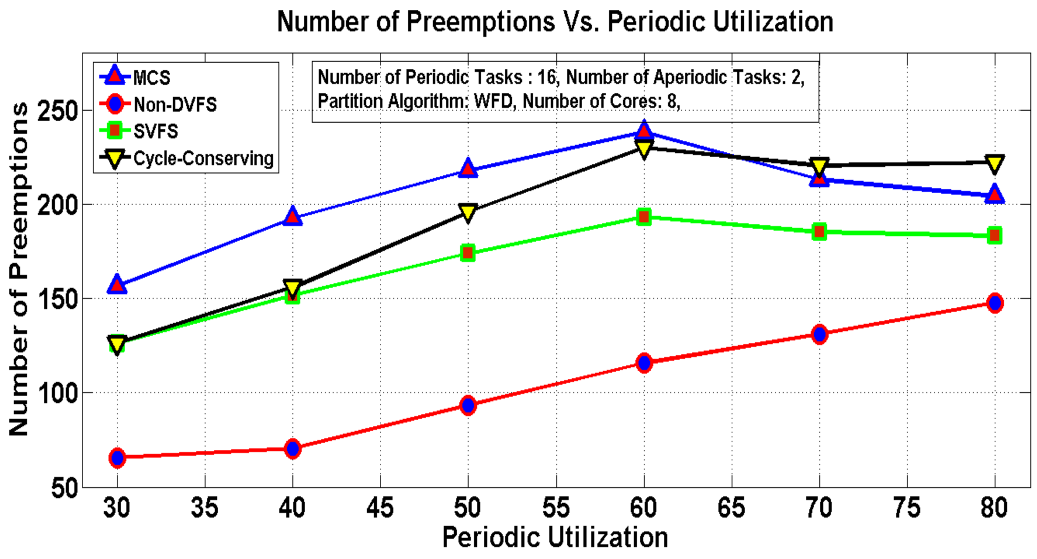


Figure 3.15: Number of Preemptions Vs. Periodic Utilization

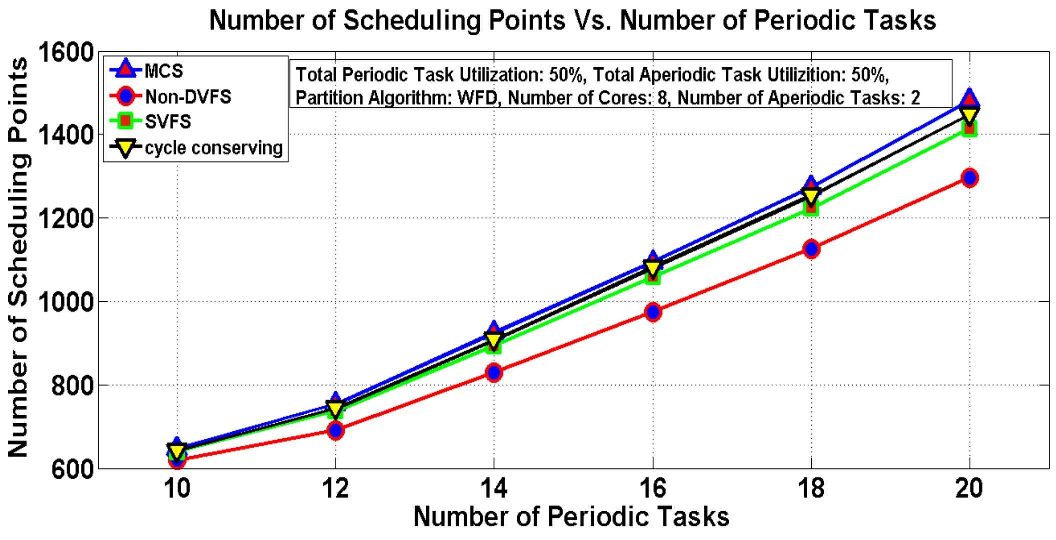


Figure 3.16: Number of Scheduling Decision Points Vs. Number of Periodic Tasks

In figures 3.17, 3.18 and 3.19, the number of migrations of aperiodic jobs is analyzed with respect to increase in periodic tasks, periodic utilization and number of processing cores. In all the three cases, the number of migrations increases with increase in each of the parameters. As periodic tasks are increased, the number of arrival of higher priority jobs increases which results in increase in preemptions of periodic and aperiodic jobs. The increase in aperiodic preemptions results in increase in migrations. In case of increase in periodic utilization, the total execution time of all the jobs over the hyper period increase which results in more number of aperiodic preemptions and migrations. It is seen from figure 3.19 that with the increase in processing cores, the opportunity of finding lightly loaded cores increases which results in increase in the number of migrations.

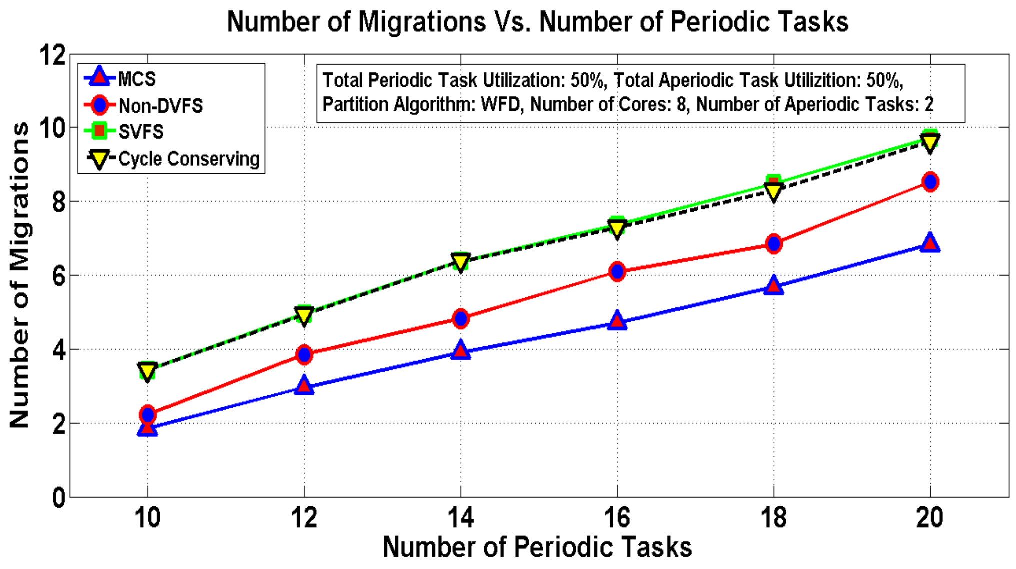


Figure 3.17: Number of Migrations Vs. Number of Periodic Tasks

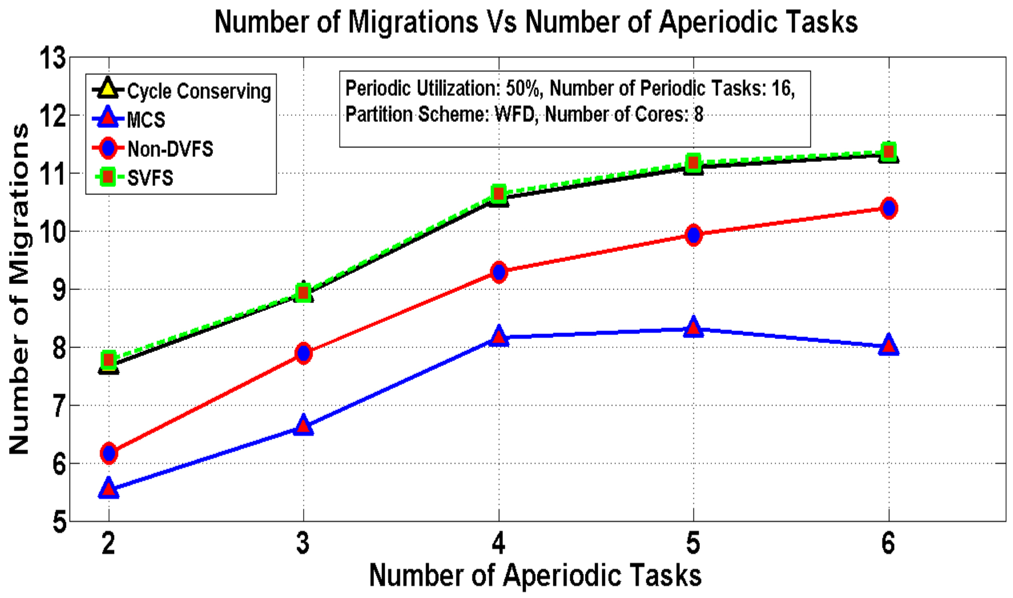


Figure 3.18: Number of Migrations Vs. Number of Aperiodic Tasks

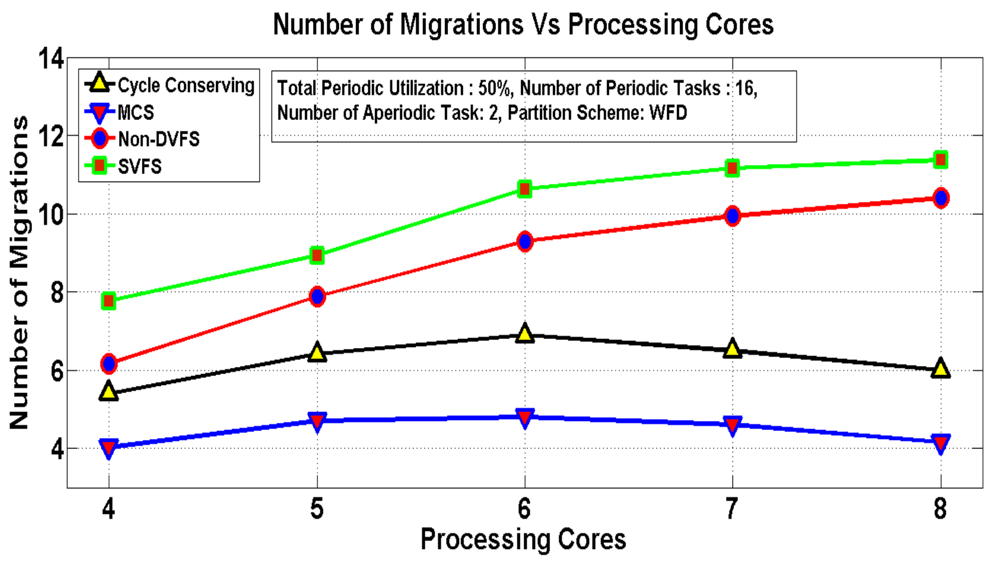


Figure 3.19: Number of Migrations Vs. Processing Cores

It can be observed from the above analysis that proposed MCS algorithm achieves more energy saving than the other algorithms at the cost of slight scheduling overhead. It also achieves comparable response time of aperiodic tasks even with increased execution time. Table 3.7 shows the percentage energy saving and percentage increase in average response time of aperiodic tasks measured using MCS algorithm with respect to other existing algorithms.

Table 3.7: Performance of the Proposed MCS Algorithm

MCS compared with	**% Energy Saving using MCS**	**% Increase in Response Time using MCS**
Non-DVFS	29.4%	2.5%
SVFS	10.1%	0.8%
Cycle Conserving Algorithm	8.9%	1.1%

3.10 Summary

Energy optimization has become an important concern in majority of the embedded systems. This chapter focused on dynamic energy optimization by using real time scheduling algorithms for mixed task sets that contain periodic as well as aperiodic tasks on uniprocessor and homogeneous multi-core processor. Energy efficient real time scheduling algorithms, EEDVFS, EE-UCS and MCS, are proposed for the optimization of dynamic energy consumption of the processors. In addition to it, the proposed algorithms also optimize the response time of aperiodic tasks. EEDVFS and EE-UCS differ in the way they schedule aperiodic tasks. EEDVFS uses DS where as EE-UCS uses TBS for scheduling aperiodic tasks. It is observed that the use of TBS improves the response time of aperiodic tasks significantly. Therefore, in case of MCS, TBS is used for aperiodic task scheduling. MCS works in two steps: task allocation and task scheduling. Static task allocation method is followed for periodic tasks while arbitrarily arriving aperiodic tasks are dynamically allocated to the least loaded processor core. The use of hybrid task allocation improves the utilization of each core in the system. The periodic and aperiodic tasks assigned to each core are independently scheduled on respective cores. The real time scheduling algorithms EDF and TBS are used to schedule the mixed task set. The proposed DVFS based energy optimization techniques show significant reduction in dynamic energy consumption.

The proposed algorithms are simulated using the proposed simulation tool STREAM and the analysis is done on various parameters such as energy consumption, aperiodic response time, preemption count, scheduling decision points, migration count etc. The behavior of the algorithms is tested on these parameters by varying number of periodic tasks, number of aperiodic tasks, number of processing cores and by increasing total utilization of the periodic tasks in a task set. For each performance metric, 100 runs were made on 100 different randomly generated task sets. On the basis of the simulation results, it is observed that, MCS performs better than all other algorithms used for comparison. It significantly reduces energy consumption as compared to non-energy aware scheduling algorithms. It gives better results than the existing energy efficient cycle conserving scheduling technique. There is little overhead in terms of preemptions, migrations and other scheduling events that consume some energy. MCS saves 29.4%, 10.1% and 8.9% energy as compared to Non-DVFS, SVFS and cycle conserving algorithm respectively over one hyper period. This states that over multiple hyper periods, MCS will show significant amount of energy saving. Similarly, In case of aperiodic tasks, percentage increase in response time by MCS with respect to other algorithms is very small and acceptable. The aperiodic response time achieved by MCS is nearly equal to the minimum possible value.

The limitation of DVFS is that if the frequency is reduced below critical speed, the static energy consumption increases and as a result, it increases the overall energy consumption. In this work, although, the frequency is not reduced below critical speed, the static energy optimization is not explicitly taken care. We have proposed a new algorithm that optimizes both dynamic and static energy consumption which is discussed in next chapter.

Chapter 4

Leakage Aware Dynamic Voltage and Frequency Scaling Based Scheduling for Multi-core Systems

This chapter discusses a real time, leakage aware dynamic voltage and frequency scaling based scheduling algorithm which reduces overall energy consumption of the processor.

4.1 Introduction

DVFS based scheduling techniques are effective in reducing dynamic energy consumption but these techniques have a major bottle neck. If the speed of execution is decreased below a certain threshold value, they result in drastic increase in sub-threshold leakage current (Lee et al., 2003; Jejurikar and Gupta, 2004). This increase in leakage current results in increase in static energy consumption which significantly increases the overall energy consumption. The proposed leakage aware DVFS based scheduling algorithm discussed in this chapter significantly reduces the dynamic as well as static energy consumption.

Jejurikar et al. (2004) suggested that the overall energy efficiency can be achieved only if the processor is running above the critical speed. Another way to achieve energy efficiency and to reduce static energy consumption is to shutdown the processor(s) whenever it is in idle state. As shutting down the processor during idle period may incur overhead, it is required to set a threshold, called break even time, to decide whether it is energy efficient to shutdown or not. For example, the break even time of a 70nm Transmeta Crusoe processor is 2 msec (Jejurikar et al., 2004). In order to stretch the idle period to reduce the number of short shutdown intervals and also reduce shutdown overhead, various procrastination techniques are in use (Jejurikar et al., 2004). Procrastination techniques delay the execution of jobs to increase the length of idle interval. Thus the overall energy optimization can be achieved only by optimizing the combined energy consumption of static and dynamic components. This in turn can be achieved by combining slowdown and shutdown techniques.

This chapter describes the proposed energy efficient scheduling algorithm named as Leakage Aware Multi-core Scheduler (LAMCS). It optimizes both dynamic and static energy consumption for a mixed task set containing periodic and aperiodic tasks on a DVFS enabled homogeneous multi-core processor platform where each core is having its own physical clock. It combines slowdown and shutdown techniques to achieve overall energy optimization. It also achieves early response time of aperiodic tasks without hampering the hard deadline of periodic tasks.

4.2 System Model

System model includes processor, task and energy models. The processor and task models are same as discussed in chapter 3 (section 3.2.2). The processor under consideration is a homogeneous multi-core processor consisting of M identical processor cores. Each core is capable of changing frequency and voltage dynamically and they have their own physical clock. The target task set includes a mix of periodic and aperiodic tasks with hard and soft deadlines respectively. The periodic tasks under consideration are highly critical in nature such that they cannot miss the deadline. The aperiodic tasks do not have hard timing constraints but they require a good average response time.

The current energy model differs from the energy model discussed in chapter 3 where we only considered the optimization of dynamic energy. In this chapter we would be measuring and optimizing overall energy consumption that includes both dynamic and static energy. The overall power consumption of a DVFS enabled CMOS based multi-core processor consist of two types of power components: dynamic power ($P_{dynamic}$) due to switching activities and static power (P_{static}) due to leakage current. $P_{dynamic}$ is a convex function of processer speed which contributes to the larger part of the total power consumption during instruction execution whereas P_{static} occurs due to different leakage sources such as sub-threshold leakage current (I_{subn}) and reverse bias junction current (I_j). In addition to dynamic and static power consumption, there is an innate power cost to keep the processor on, which can be denoted as P_{on}. Certain processor components such as Phase Locked Loops (PLL) circuitry, I/O subsystems etc., consumes power even if no work is carried out by the processor. The power consumption of such components adds up to a significant portion of the total power consumption. Considering these three sources of power consumptions, the total power consumption can be stated as:

$$P_{total} = P_{dynamic} + P_{static} + P_{on} \tag{4.1}$$

$$P_{dynamic} = C_{ef} \times {V_{dd}}^{2} \times f \tag{4.2}$$

$$P_{static} = (V_{dd} \times I_{subn} + |V_{bs}| \times I_j) \times L_g \tag{4.3}$$

$$I_{subn} = K_3 \times e^{K_4 \times V_{dd}} \times e^{K_5 \times V_{bs}} \tag{4.4}$$

where V_{dd} and f are the supply voltage and maximum frequency of the processor. The description and values of the constants in the above equations are given in table 4.1. The values of these constants are based on Transmeta Crusoe processor, scaled at 70nm technology (Jejurikar et al 2004). The maximum voltage and frequency of this processor are 1.0 V and 3.1GHz respectively.

To assess the potential of DVFS and leakage aware scheduling, the total dynamic and static energy consumption can be measured per cycle for different values of supply voltages as follows:

$$E_{total} = E_{dynamic} + E_{static} + E_{on} \tag{4.5}$$

where,

$$E_{dynamic} = C_{ef} \times {V_{dd}}^{2} \tag{4.6}$$

$$E_{static} = (V_{dd} \times I_{subn} + |V_{bs}| \times I_j) \times L_g \times f^{-1} \tag{4.7}$$

$$E_{on} = P_{on} \times f^{-1} \tag{4.8}$$

Table 4.1: 70nm Technology Constants

Constant	Value
Effective switching capacitance (C_{ef})	0.43×10^{-9}
Body bias voltage (V_{bs})	-0.7V
Technology constant (K_3)	5.38×10^{-7}
Technology constant (K_4)	1.83
Technology constant (K_5)	4.19
Reverse bias junction current (I_j)	4.8×10^{-10}
Number of devices in the circuit (L_g)	4×10^{6}

According to Eq. (4.6) and Eq. (4.7), even though it is observed that there is quadratic and linear dependence of dynamic and static energy respectively on supply voltage, it does not result in energy saving with decrease in voltage after certain limit. This is because the static energy consumption increases if the processor speed decreases below a threshold. This threshold is called critical speed. If the processor runs below this speed, the static power consumed due to leakage current nullifies the gains of dynamic voltage scaling and as a result the overall energy consumption increases. The critical speed of Transmeta Crusoe processor is 1.26 GHz which is 41% of maximum speed. The voltage corresponding to critical speed is 0.7 V. The experimentation carried out in this paper takes care of critical speed and does not slow down the processor below this speed. The details of frequency and voltage ranges are given in table 3.5 of chapter 3 (Section 3.7.5).

4.3 Shutdown Overhead

Dynamic energy is minimized using DVFS technique while static energy is minimized by putting the processor in shut down mode when it is in idle state. However, putting the processor in shut down state and then waking it up incurs some overhead because the processor loses temporal data stored in various forms of memory such as registers, caches, TLBs etc. Thus before shut down, all registers must be saved and dirty cache lines must be written back to the memory and upon wake up all the saved data must be retrieved back to registers, cache lines etc. This results in additional memory accesses and hence additional energy consumption. Therefore, in order to decide whether to shut down the processor or not, threshold of idle interval is computed based on the idle state power consumption and shut down overhead. If the idle interval is less than threshold idle interval then it is not energy efficient to shut down the processor. The exact length of threshold interval varies for different processor architectures. For the Transmeta Crusoe processor, the value of idle threshold interval is 2 ms (Jejurikar et al., 2004).

4.4 Proposed Leakage Aware Multi-core Scheduling Algorithm (LAMCS)

LAMCS is an energy efficient real-time mixed task set multi-core scheduling algorithm. LAMCS addresses optimization of both dynamic and static energy for multi-core processors. The algorithm is organized into three sub-parts: (1) Task Allocation (2) Voltage and Frequency Selection and (3) Procrastination and Shutdown.

4.4.1 Task Allocation

The suggested solution uses both partitioned and global methods for task allocation. Since periodic tasks are highly critical and are not allowed to miss deadlines, they are partitioned among multiple cores. Once allocated to the respective cores, the jobs corresponding to the tasks are executed on that core and are not allowed to migrate as migration incurs additional overhead which may result in deadline misses. Task partitioning is done using Bin Packing heuristics (Gray and Johnson, 1979) like First Fit Decreasing (FFD), Worst Fit Decreasing (WFD). The partitioning divides the task set into M subsets where M is the number of cores in a multi-core system. Let each subset be denoted as S_i where i denotes the i^{th} core among M cores. Hence, S, the set of M subsets of periodic tasks, is defined as $\{S_0, S_1, ..., S_{M-1}\}$. The algorithm maintains a set of M job queues, $JQ = \{JQ_0, JQ_1, JQ_2, ..., JQ_{M-1}\}$, one per core C_i. The jobs of the tasks belonging to subset S_i are inserted in job queue JQ_i as soon as they are ready for execution. Aperiodic job allocation is done using global approach where it can be assigned to any core upon its arrival and can be migrated to other cores upon preemption. It is better to use global assignment as aperiodic tasks have soft deadlines and migration to lightly loaded core improves the response time of the task.

4.4.2 Voltage and Frequency Selection

At each core, jobs of the periodic and aperiodic tasks are scheduled. Periodic jobs are scheduled using EDF, which is an optimal uniprocessor scheduling algorithm (Liu, 2008). Aperiodic jobs are scheduled using TBS which offers simple implementation and better performance than other existing aperiodic servers (Spuri and Buttazzo, 1996). The scheduling decisions are made either on completion of an executing job or on arrival of a new job. At each scheduling time t on a core C_i, Dynamic utilization $U_{dyn}(t)$ for periodic job P_J with a deadline D_{PJ} is computed as:

$$U_{dyn}(t) = U_{past}(t) + U_{future\_low}(t) + U_{future\_high}(t) \tag{4.9}$$

where U_{past} *(t)* denotes the sum of the remaining utilization of the periodic jobs which were released before time *t* and $U_{future\_low}(t)$ and $U_{future\_high}(t)$ denote the sum of share of processor utilization of the periodic jobs that will be released during the interval [t, D_{PJ}). The subscripts *future_low* and *future_high* in equation 4.9 indicate that the jobs to be released are of lower priority and higher priority than the scheduled job respectively. The expressions to calculate U_{past} *(t)*, $U_{future\_low}(t)$ and $U_{future\_high}(t)$ are given as below:

$$U_{past}(t) = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t, D_{PJ}] \wedge preempted\ J_i)} \frac{c_{i\_rem}}{D_{PJ} - t} \tag{4.10}$$

$$U_{future\_low}(t) = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t, D_{PJ}] \wedge\, !preempted\ J_i \wedge Di \geq D_{PJ})} \frac{c_i}{pi} \times (D_{PJ} - a_i) \tag{4.11}$$

$$U_{future\_high}(t) = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t, D_{PJ}] \wedge\, !preempted\ J_i \wedge Di < D_{PJ})} c_i \tag{4.12}$$

where J_i is i^{th} periodic job arriving between [t,D_{PJ}), c_{i_rem} is the remaining worst case execution time of the preempted job and *t* is the current time. Based on the current dynamic utilization of the processor core, a matching smallest frequency f_{opt} and its corresponding voltage v_{opt} is selected such that U_{dyn} is less than or equal to scaling factor α where $\alpha = f_{opt}/f_{max}$, with $f_{opt} \in \{f_1, f_2, ..f_{max} \mid f_1 < f_2 < ... < f_{max}\}$. The algorithm 4.1 depicts the frequency selection method of the proposed LAMCS algorithm. The proposed method of calculating dynamic utilization gives the exact available utilization at any scheduling point and assigns an optimal frequency to the scheduled job. Periodic job runs at optimal frequency if there is no aperiodic job in its job queue. In presence of an aperiodic job, it runs at maximum frequency as TBS uses the remaining processor utilization (Eq. 4.15) other than periodic tasks utilization for the execution of aperiodic jobs leaving no scope for DVFS.

Algorithm 4.1: Frequency Selection of Proposed LAMCS Algorithm

```
Select_Frequency
Begin
        Calculate $U_{dyn}$ from eq. 4.9, 4.10, 4.11 and 4.12
        select smallest freq $f_{opt} \in \{f_1, f_2, ..f_{max} \mid f_1 < f_2 < ... < f_{max} \land f_1 > f_{critical}\}$
                such that $U_{dyn} <= f_{opt}/f_{max}$
        Returns optimal frequency $f_{opt}$
End
```

Upon arrival of an aperiodic task, its virtual deadline is calculated using TBS algorithm based on the current utilization of each core. The virtual deadline is denoted as v_{k_i} for aperiodic job A_k on core C_i and is defined as follows:

$$v_{k\_i} = Max\{a_k, v_{k-1\_i}\} + \frac{c_k}{U_{srv\_i}} \tag{4.13}$$

$$v_{k\_i} = Max\{WT_k, v_{k-1\_i}\} + \frac{c_k}{U_{srv\_i}} \tag{4.14}$$

where, $U_{srv\_i} = 1 - U_{dyn}$ (4.15)

U_{srv_i} is the utilization of TBS on core C_i at the arrival time a_k of aperiodic job A_k and WT_k is the wake-up time of core C_i when C_i is in shut down mode during arrival of A_k. Eq. (4.13) is used when the core is in running or idle state while Eq. (4.14) is used when the core is in shut down state. After calculating the virtual deadlines on each core, the aperiodic job is assigned to the processor core on which it finds earliest virtual deadline amongst all the cores as shown:

$$v_{min\_index} = Min_{0 \leq i \leq M-1}\{v_{k\_i}\} \tag{4.16}$$

where *min_index* denotes the index of a processor core that gives earliest virtual deadline among all the cores. Virtual deadline calculations and selection of core are done centrally by the aperiodic controller and has very small overhead. This aperiodic controller has the knowledge of current utilizations of all the cores.

4.4.3 Procrastination and Shutdown

When the job queue of a core C_i is empty, procrastination interval (PI_i) is calculated and compared with the threshold shutdown interval. If it is greater than or equal to the threshold time interval, the processor core is put in shutdown mode till the timer is exhausted. Algorithm 4.2 shows the procrastination algorithm and describes calculation of procrastination interval. It postpones the execution of periodic jobs that are arriving during shutdown period to extend the shutdown interval. Let AT_{Ji} be the arrival time of a periodic job J_i on core C_i which is arriving after the idle interval has started. The procrastination interval (Z_j) is calculated by considering all the jobs arriving in near future. If this interval is greater than the shutdown threshold, then this will act as the next wake up time (WT). The processor backs all the relevant data before shutting down the processor till WT. Procrastination interval Z_j is calculated as follows:

$$Z_J(t) = 1 - (U_{future\_low}(t) + U_{future\_high}(t)) \tag{4.17}$$

The wake up time value is chosen to ensure the timely completion of postponed jobs which will be executed after the processor core wakes up. The first job after wakeup is always executed at maximum frequency as the procrastination timer is calculated based on the *wcet* of jobs at maximum frequency. The complete algorithm, LAMCS, is shown in algorithm 4.3. Figure 4.1 shows the detail flow of LAMCS in a flow chart by highlighting the sub-modules. The notations used in the flowchart are same as in algorithm LAMCS.

Algorithm 4.2: Procrastination Interval Calculation in Proposed LAMCS Algorithm

```
Procrastination_Interval (Core Ci, Current Time loc_time)
Begin
    WT = HP
    $J_{next}$ = next periodic job which will arrive during idle interval;
    while ($AT_{Jnext}$ < WT)
    Begin
        compute Z $_{Jnext}$ for $J_{next}$ using equation 4.17
        WT = min (WT, $AT_{Jnext} + Z_{Jnext}$)
        $J_{next}$ = next periodic job
    End
    set PI = WT - loc_time
    return PI
End
```

Algorithm 4.3: Proposed Leakage Aware Multi-core Scheduling Algorithm (LAMCS)

Pre-Condition: S_i, set of periodic tasks belonging to core C_i after partition.
Post-Condition: Feasible schedule if exists; FAIL otherwise.
Waking Up Condition: Core C_i wakes up when procrastination timer exhausts.

```
LAMCS
Begin
    Join all pending jobs corresponding to core $C_i$ in $JQ_i$
    if Empty ($JQ_i$) then
        $PI_i$ = Procrastination_Interval ($C_i$,t)
        if $PI_i$ >= threshold interval then
            set Procrastination Timer to $PI_i$
            Put core $C_i$ in shutdown state
            Return
        else
            Core $C_i$ remain idle
            Return
        end if
    end if
    if Curr.$JQ_i$.RemainingTime == 0 then
        Curr.$JQ_i$ = Head.$JQ_i$
    else if Head.$JQ_i$.Priority > Curr.$JQ_i$.Priority then
        if Curr.$JQ_i$ is periodic then
            preempt and insert Curr.$JQ_i$ in $JQ_i$
        else
            find core with earliest virtual deadline using eq. 4.13, 4.14 and
                                                                    4.17
            if min_index != i then
                preempt and migrate Curr.$JQ_i$ to core $C_{min\_index}$
            else
                preempt and insert Curr.$JQ_i$ in $JQ_i$
            end if
        end if
        Curr.$JQ_i$ = Head.$JQ_i$
    end if
    if ! DVFS OR $JQ_i$ has at least one aperiodic job OR Curr.$JQ_i$ is aperiodic job OR
    $PI_i$ is exhausted then
        execute Curr.$JQ_i$ till Min(Curr.$JQ_i$.Remaining Time, Arrival of next job)
        at $f_{max}$
    else
        $f_{opt}$ = Select_Frequency($C_i$)
        execute Curr.$JQ_i$ till Min(Curr.$JQ_i$.Remaining Time / $\alpha$, Arrival of next
        job) at $f_{opt}$
    end if
End
```

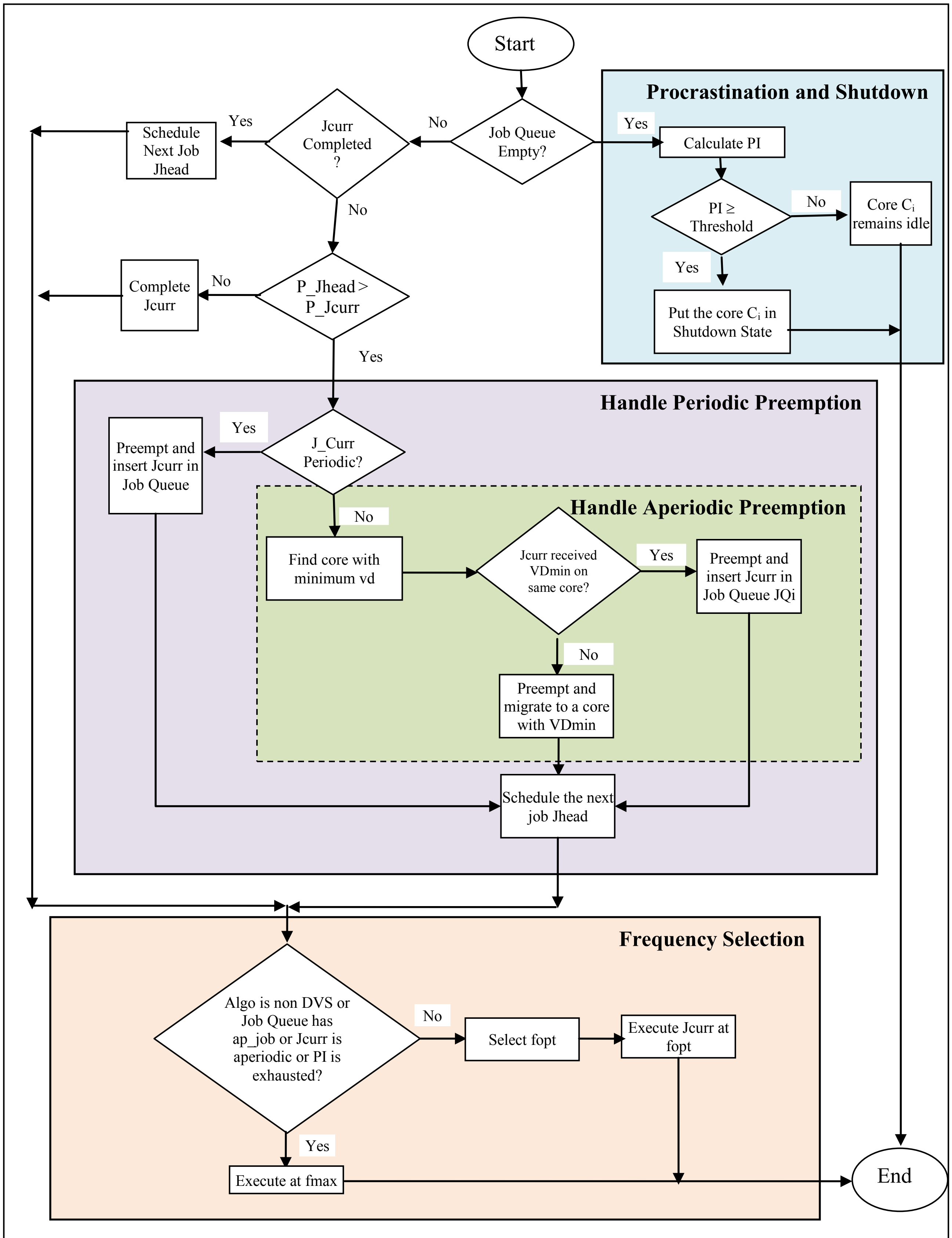

Start
Procrastination and Shutdown
Job Queue Empty?
Yes
No
Calculate PI
Jcurr Completed ?
Yes
Schedule Next Job Jhead
PI ≥ Threshold
No
Core Ci remains idle
Yes
Put the core Ci in Shutdown State
No
P_Jhead > P_Jcurr
No
Complete Jcurr
Yes
Handle Periodic Preemption
J_Curr Periodic?
Yes
Preempt and insert Jcurr in Job Queue
No
Handle Aperiodic Preemption
Find core with minimum vd
Jcurr received VDmin on same core?
Yes
Preempt and insert Jcurr in Job Queue JQi
No
Preempt and migrate to a core with VDmin
Schedule the next job Jhead
Frequency Selection
Algo is non DVS or Job Queue has ap_job or Jcurr is aperiodic or PI is exhausted?
No
Select fopt
Execute Jcurr at fopt
Yes
Execute at fmax
End

Figure 4.1: Flowchart of Proposed LAMCS Algorithm

4.5 Schedule of Sample Task Set using LAMCS Algorithm

This section describes the proposed algorithm, LAMCS by generating a schedule of a mixed task set on two cores. Table 4.2 shows the attributes of a task set consisting of three periodic tasks (T_0,T_1, and T_2) and 2 aperiodic tasks (A_0 and A_1) on a dual core processor. Hyper period (HP) of the task set is 50. A periodic job $J_{<mn>}$ is denoted as $(n+1)^{th}$ job of task T_m. The tasks are allocated on the basis of WFD partitioning scheme. The resultant schedule while executing with LAMCS on Core 0 and Core 1 is shown in tables 4.3 and 4.4 respectively. Each row in table 4.3 and table 4.4 is dedicated to either arrival or completion of a job. At each scheduling point, each row in these tables show which job has arrived, started, migrated, completed or has been preempted. It shows the value of dynamic utilization of the core at a scheduling event. Based on the utilization, selected optimal frequency and scaled *aet* is also shown. In case of aperiodic job, its virtual deadline is shown in the respective column. In addition to all these parameters, state of the processor core (running/shutdown/wakeup) and value of the procrastination timer are mentioned which are used to defer execution of a periodic job.

As can be seen in table 4.3, on Core 0, at time 0, J_{10} is ready for execution at a current utilization of 0.6. Therefore, J_{10} is scheduled and executed at 60% of maximum frequency. At time 25, upon arrival of A_1, it is assigned to Core 0 since the virtual deadline calculated for Core 0 is earlier than that calculated for Core 1. At time 25, J_{10} is preempted as the deadline of A_1 is earlier than J_{10} and A_1 is scheduled. At time 30, when A_1 finishes, the current utilization reduces to 0.75. As a result, J_{10} executes at 80% of maximum frequency and finishes at time 40.

Similarly, as seen in table 4.4, on Core 1, at time 0, J_{00} and J_{20} have arrived. According to EDF scheduling policy, J_{20} is selected for execution at optimal frequency, 60% of f_{max}. Upon its completion at time 1.7, dynamic utilization is calculated as 0.57 using Eq. (4.9), optimal frequency closest to utilization is selected as 60% of f_{max} and the next ready job J_{00} is selected for execution at this frequency. At time 6.7, J_{00} completes and processor core 1 is put in shutdown mode for 11.3 units of time. At time 8, upon arrival of aperiodic job A_0, virtual deadline is calculated for A_0 on both the cores using Eq. (4.13) and (4.14). A_0 is assigned to Core1 since it gets early virtual deadline on

Core1. As Core1 is in shutdown state, A_0 waits in job queue till the processor wakes up. It is then scheduled in priority order. At time 18, upon wake up, J_{21} is scheduled which completes at 19.2 and A_0 gets the chance to execute on Core 1. A_0 is preempted at time 20 due to arrival of higher priority job J_{22} which executes for 0.2 units of time and finishes at time 19.4. A_0 resumes execution and preempts at time 30. At time 30, J_{01} and J_{23} which arrived at time 25 and 30 respectively are waiting in the ready queue. As J_{23} has earliest deadline among the three job (J_{23}, A_0, J_{01}) in ready queue, J_{23} is scheduled and executed for 1.6 units of time. At time 31.6, A_0 resumes and finishes at time 37.2. It should be noted that in presence of aperiodic job, as the total utilization is 100%, all the jobs are executed at maximum frequency. Upon completion of aperiodic job A_0, the current utilization is 0.94, therefore, J_{01}, is executed at maximum frequency. But upon completion of J_{01} at time 44.2, the current utilization drastically reduces to 0.34. As a result, the next periodic job J_{24} executes at 40% of maximum frequency. The entire schedule till the hyper period is shown in table 4.3 and table 4.4 for Core0 and Core1 respectively. Figure 4.2 shows the timing diagram of the entire schedule for the example task set. The horizontal axis shows the time line from time 0 till the hyper period 50 and the vertical axis shows the execution frequency.

Table 4.2: Mixed Task Set (LAMCS)

Tasks	**Arrival**	**Period**	***wcet***	**Deadline**	**Core # (Partitioning Scheme: WFD)**	**U_{wc}**	**# of jobs in a HP**	***aet* of jobs**
T_0	0	25	10	25	Core 1	0.4	2	3, 7
T_1	0	50	30	50	Core 0	0.6	1	23
T_2	0	10	2	10	Core 1	0.2	5	1, 1.2, 1.4, 1.6, 1.8
A_0	8	-	15	-		-	-	15
A_1	25	-	5	-		-	-	5

Table 4.3: Schedule on Core 0 using Proposed LAMCS Algorithm

Time	R/M	S/RP/C/P	Comp	Preempt (RAET)	UT(R/C/W)	Freq (%)	WCET, AET/RAET	Scaled AET	VD	CPU State	PT
0	J_{10}/-	J_{10}/-/-/-	-	-	0.6/-	60	30,23 / -	38.3	-	Running	-
8	A_0/-	-/-/J_{10}/-	-	-	-	-	15,15/-	-	45.5(not Allocated)	Running	-
20	A_0 preempted on Core1	-/-/J_{10}/-	-	-	-	-	-	-	55.3 (not Migrated)	-	-
25	A_1/-	A1/-/-/-	-	J_{10}(8)	1/-/-	100	5,5/-	5	37.5	Running	-
30	-/-	-/J_{10}/-/-	A_1	-	-/0.75/-	80	-,-/8	10	-	Running	-
40	-/-	-/-/-/-	J_{10}	-	-	-	-	-	-	Shutdown	10.0

R/M - Release/Migration, S/RP/C/P - Started/Resumed after Preemption/Continued, Comp - Completed, Preempt (RAET) - Preempted (Remaining Actual Execution Time at maximum frequency), UT(R/C) - Dynamic Core Utilization (upon Release/upon Completion), Freq (%) - Normalized Frequency in %, WCET, AET/RAET - Worst Case Execution Time of released job, Actual Execution Time/Remaining Actual Execution Time at maximum frequency, Scaled AET - Actual Execution Time at scaled frequency, VD - Virtual Deadline of aperiodic task, PT - Procrastination Timer

Table 4.4: Schedule on Core 1 using Proposed LAMCS Algorithm

Time	R/M	S/RP/C/P	Comp	Preempt (RAET)	UT(R/C/W)	Freq (%)	WCET, AET/RAET	Scaled AET	VD	CPU State	PT
0	J_{00},J_{20}/-	J_{20}/-/-/-	-	-	0.6/-	60	2,1/-	1.7	-	Running	-
1.7	-	J_{00}/-/-/-	J_{20}	-	-/0.57	60	10,3/-	5	-	Running	-
6.7	-	-/-/-/-	J_{00}	-	-	-	-	-	-	Shutdown	11.3
8.0	A_0/-	-/-/-/A_0	-	-	1/-	100	15,15/-	15	44.25	Shutdown	-
10	J_{21}/-	-/-/-/J_{21}	-	-	-	-	-	-	-	Shutdown	-
18	-/-	J_{21}/-/-/-	-	-	-/-/1	100	2,1.2/-	1.2	-	WakeUp	-
19.2	-/-	A_0/-/-/-	J_{21}	-	-/1/-	100	15,15/-	15	44.25	Running	-
20.0	J_{22}/-	J_{22}/-/-/-	-	A_0(14.2)	1/-/-	100	2,1.4/-	1.4	-	Running	-
21.4	-/-	-/A_0/-/-	J_{22}	-	-/1/-	100	-,-/14.2	14.2	44.25	Running	-
25.0	J_{01},A_1/-	-/-/A_0/-	-	-	1/-/-	100	-,-/10.6	10.6	VD_{A1} = 55.6, not allocated	Running	-
30.0	J_{23}/-	J_{23}/-/-/-	-	A_0(5.6)	1/-/-	100	2,1.6/-	1.6	-	Running	-
31.6	-/-	-/A_0/-/-	J_{23}	-	-/1/-	100	-,-/5.6	5.6	44.2	Running	-
37.2	-/-	J_{01}/-/-/-	A_0	-	-/0.94/-	100	10,7/-	7	-	Running	-
40.0	J_{24}/-	-/-/J_{01}/-	-	-	-/-/-	100	-	-	-	Running	-
44.2	-/-	J_{24}/-/-/-	J_{01}	-	-/0.34/-	40	2,1.6/-	4.5	-	Running	-
48.7	-/-	-/-/-/-	J_{24}	-	-/-/-	-	-	-	-	Idle	-

R/M - Release/Migration, S/RP/C/P - Started/Resumed after Preemption/Continued, Comp - Completed, Preempt (RAET) - Preempted (Remaining Actual Execution Time at maximum frequency), UT(R/C) - Dynamic Core Utilization (upon Release/upon Completion), Freq (%) - Normalized Frequency in %, WCET, AET/RAET - Worst Case Execution Time of released job, Actual Execution Time/Remaining Actual Execution Time at maximum frequency, Scaled AET - Actual Execution Time at scaled frequency, VD - Virtual Deadline of aperiodic task, PT - Procrastination Timer

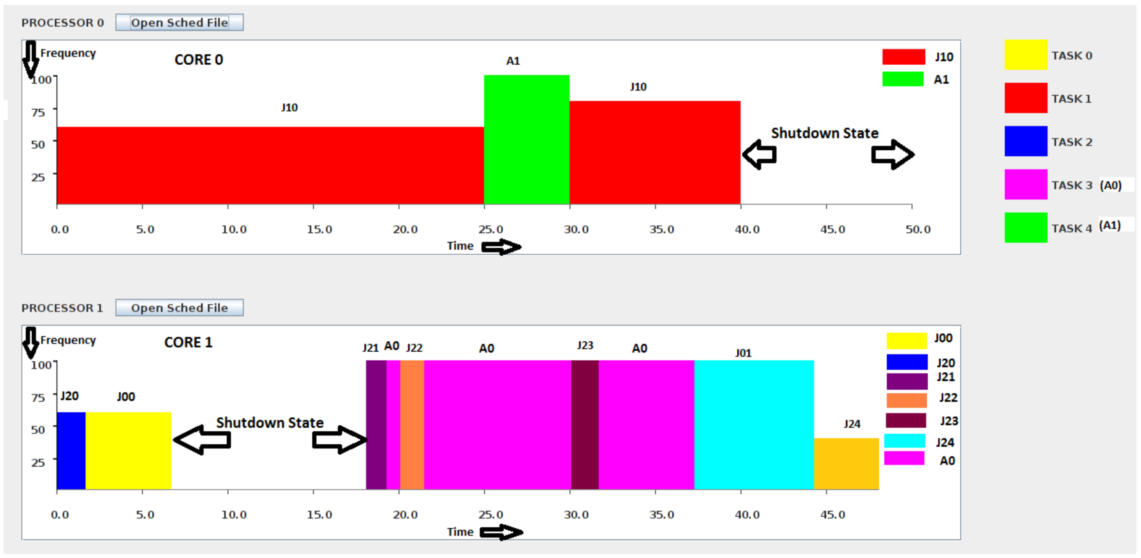


Figure 4.2: Timing Diagram for the schedule on Core 0 and Core 1

4.6 Correctness Proof and Schedulability of LAMCS Algorithm

Theorem 1: A system of independent preemptable mixed tasks containing N periodic and M aperiodic tasks, where periodic tasks have implicit deadlines and are in-phase, aperiodic tasks have soft deadlines and arrive arbitrarily, is schedulable on K cores according to LAMCS if the periodic tasks are partitionable among K processor cores using Bin Packing heuristics, and the total utilization of periodic tasks and aperiodic tasks is less than K.

Proof: LAMCS follows worst fit decreasing and first fit decreasing partitioning heuristics which guarantee the assignment of tasks among K cores. Since the Bin Packing partitioning heuristics provides the utilization bound of 66% (Anderson and Tovar, 2006), the remaining utilization of each processor core is used for executing aperiodic tasks thereby increasing the schedulability of the task set.

LAMCS follows EDF scheduling policy for scheduling periodic tasks on each processor core independently. On the other hand, TBS scheduling policy is used for scheduling aperiodic tasks along with periodic tasks where the utilization of TBS server is equal to the remaining utilization of the processor core other than the assigned utilization of periodic tasks.

As EDF along with TBS scheduling policies are optimal (Liu, 2008) for uniprocessors, bin packing heuristic allocates periodic tasks on multiple cores with utilization bound of 66% and aperiodic tasks use the remaining utilization of processor cores, LAMCS produces a feasible schedule on multi-core processor and increases the utilization bound of the processor cores.

Theorem 2: In a system of independent preemptable mixed tasks consisting of periodic and aperiodic tasks where periodic jobs follow EDF schedule and aperiodic jobs follow TBS schedule, the dynamic energy optimization guarantees least idle time, static energy and dynamic energy with maximum shutdown duration achieved by DVFS and procrastination.

Proof: The algorithm LAMCS selects the optimal frequency and voltage at each scheduling point on the basis of dynamic utilization of the processor core at that scheduling point. The dynamic utilization, frequency and voltage are calculated on the basis of current load of the processor core which results in minimum idle intervals. The correctness of dynamic utilization of a processor core is proved by using mathematical induction in corollary 1.

When the processor core is idle, LAMCS computes the shutdown interval by using Eq. (4.17). The shutdown interval is extended by using dynamic procrastination resulting in achieving maximum shutdown interval. The correctness of shutdown interval with dynamic procrastination is proved by using mathematical induction in corollary 2.

Corollary 1: When a task set *T* contains *N* periodic tasks and *M* aperiodic tasks where *N* ≥ 2 and *M* > 0, the dynamic utilization of the processer at any scheduling time t is given by

$$U_{dyn}(t) = U_{past}(t) + U_{future\_low}(t) + U_{future\_high}(t)$$

where (from Eq. 4.10, 4.11 and 4.12),

$$U_{past}(t) = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t, D_{PJ}] \wedge preempted\ J_i)} \frac{c_{i\_rem}}{D_{PJ}-t}$$

$$U_{future\_low}(t) = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t, D_{PJ}] \wedge\ !preempted\ J_i \wedge Di \geq D_{PJ})} \frac{c_i}{pi} \times (D_{PJ} - a_i)$$

$$U_{future\_high}(t) = \sum_{(\forall J_i \mid J_i \in S_i \wedge J_i \in (t, D_{PJ}] \wedge\ !preempted\ J_i \wedge Di < D_{PJ})} c_i$$

Proof:

Assume T contains $(N+M)$ tasks where $N \geq 2$ and $M > 0$. The dynamic utilization stated in the above equations is proved by using mathematical induction as follows. The mathematical induction is based on the number of jobs present in the ready queue at any scheduling point.

Basic Step: At any scheduling time t, if there are no lower priority jobs waiting in the job queue other than scheduled periodic job PJ and no job is released during interval $[t,D_{PJ})$ then

$$U_{past}(t) = U_{PJ} \tag{4.18}$$

$$U_{future\_low}(t) = 0 \tag{4.19}$$

$$U_{future\_high}(t) = 0 \tag{4.20}$$

Therefore, dynamic utilization of the core C_i is equal to worst case utilization of the scheduled periodic job PJ.

Inductive Step: At any scheduling time t, Let j be the number of lower priority jobs waiting in the job queue including scheduled job, k be the number of jobs arriving during interval $[t,D_{PJ})$ which are lower priority than PJ and l be the number of jobs arriving during interval $[t,D_{PJ})$ which are higher priority than job PJ. In this case, dynamic utilization can be stated as follows:

$$U_{past} = \sum_{i=1}^{j} \frac{c_{i\_rem}}{Dpj-t} \tag{4.21}$$

$$U_{future\_low} = \sum_{i=1}^{k} \frac{ci}{pi} \times (D_{PJ} - a_i) \tag{4.22}$$

$$U_{future\_high} = \sum_{i=1}^{l} c_i \tag{4.23}$$

Therefore, dynamic utilization at any scheduling time t will be the sum of U_{past}, U_{future_low} and U_{future_high}.

Corollary 2: For a core C_i, at time t, if JQ_i is empty and J is the periodic job which arrives after time t, the calculation of procrastination interval is the highest possible without missing any deadlines.

Proof: The corollary is proved using mathematical induction as follows:

Basic Step: At a time instance t, when the job queue JQ_i is idle and periodic job J arrives after time t, assuming no job will arrive during interval $[a_J,D_J)$, then the procrastination interval Z_J (for core C_i) can be stated as:

$$Z_J = 1 - \frac{c_J}{p_J} \tag{4.24}$$

Therefore, the execution of job *J* can be procrastinated for Z_J units of time.

Inductive Step: At a time instance *t*, when the job queue JQ_i is idle and periodic job *J* arrives after time *t*, let us assume that *k* lower priority and *l* higher priority jobs will arrive during the interval $[a_J,D_J)$. The procrastination interval Z_J (for core C_i) in this case can be stated as:

$$Z_J = 1 - (\sum_{i=1}^{k} \frac{ci}{pi} \times (D_J - a_i) + \sum_{i=1}^{l} c_i) \tag{4.25}$$

The above equation procrastinates the execution of periodic job *J* by taking care of the share of each job arriving during the interval $[a_J,D_J)$ and ensures the deadline of job *J*.

Theorem 3: In a multi-core system where independent preemptable periodic tasks are assigned and scheduled independently on different cores using LAMCS with DVFS and DPM and aperiodic jobs are globally scheduled, the response time of aperiodic jobs are not affected due to voltage and frequency scaling.

Proof: LAMCS schedules aperiodic jobs globally to any of the cores on multi-core system on the basis of virtual deadline that aperiodic job receives on each core. It assigns the job to a core which gives early virtual deadline. The correctness of virtual deadline computation for an aperiodic job is given in corollary 3. All aperiodic jobs execute in maximum voltage and frequency which offers minimum response time.

Corollary 3: In a system containing *K* processor cores, *N* periodic tasks and *M* aperiodic tasks, the virtual deadline for an aperiodic job is computed as follows (From Eq. (4.13) and (4.14)):

$$v_{k\_i} = \mathrm{Max}\{a_k, v_{k-1\_i}\} + \frac{c_k}{U_{srv\_i}}$$

$$v_{k\_i} = \mathrm{Max}\{WT_k, v_{k-1\_i}\} + \frac{c_k}{U_{srv\_i}}$$

The virtual deadline of any aperiodic job depends on the TBS utilization. For example, in a dual core system, if server utilization of core 1 is greater than server utilization of core 2, then core 1 provides early virtual deadline than core 2. From Eq. (4.15), it is observed that server utilization depends on core's dynamic utilization; the correctness of virtual deadline calculation can be proved by corollary 1.

4.7 Algorithmic Complexity of LAMCS Algorithm

The algorithmic complexity of LAMCS is the product of the number of scheduling decision points and the complexity per scheduling decision. Given a task set S with hyper period H, the algorithmic complexity of LAMCS is

$$T_{LAMCS}(S,H) = Decisions\ (LAMCS, S, H) * C_{decision}\ (LAMCS, S, H) \quad (4.26)$$

$$Decisions\ (LAMCS, S, H) = N(A_p + A_a) + NC_1 + NC_2 + NP \quad (4.27)$$

Where *Decisions* is the number of decision points to be made by the algorithm and $C_{decision}$ is the time complexity of a single decision. *Decisions* can be divided into five types of decisions points: number of periodic job arrivals $N\ (A_p)$ and number of aperiodic job arrivals $N\ (A_a)$, number of completion points (NC_1) when job queue is non-empty, number of completion points (NC_2) when job queue is empty and number of aperiodic preemption points (NP). Time complexity of LAMCS can be derived as follows:

$$T_{LAMCS}(S,H) = O(N(A_p + A_a) * (T \log_2 T)) + O(NC_1 *(K + P)) + O(NP * T\log_2 T) + O(L) + O(M) + O(NC_2 * K^2) \quad (4.28)$$

where T is the number of tasks in a task set S. The time required for insertion in job queues upon job arrival is $O(T \log_2 T)$. The constant K is the number of jobs appearing in a time interval [*t, deadline of a scheduled job*) at any scheduling time t and P is the number of lower priority jobs waiting in the job queue at time t. The number of time units required to calculate dynamic utilization at any completion time when the job queue is non-empty are (K + P). In practice, the value of K and P are very small as compared to total jobs. On preemption, the aperiodic jobs are inserted in the job queue of one of the cores. This additional insertion time is $O\ (NP * T\log_2 T)$. The time complexity of the selection of matching frequency level from the L discrete frequency levels is O (L). O (M) is the constant amount of required to select a processor core which provides earlier virtual deadline to a scheduled aperiodic job where M is the number of processor cores. The time required for taking a shutdown decision is $O\ (K^2)$ as described in equations 4.11, 4.12 and 4.17.

4.8 Energy Calculations for Proposed LAMCS Algorithm

This section presents the energy calculations of the experiments carried out in this work. The overall energy consumption includes energy consumed during task execution (i.e. dynamic energy), scheduling events such as arrival, completion, migration, slowdown, shut down and wakeup. Energy consumed during task execution, E_{exec_energy}, is calculated at various intervals over the hyper period at different frequencies and voltages. Energy consumed in performing scheduling events is considered as scheduler overheads $E_{sched\_overhead}$. For the calculation of $E_{sched\_overhead}$, it is required to know the number of job arrivals, completions, preemptions, migrations and scheduling decisions throughout the hyper period for each task set. The static energy E_{static}, shutdown energy $E_{shutdown}$ and shutdown overhead energy $E_{shutdown\_overhead}$ consumption per unit time are considered as per Transmeta Crusoe processor. The overall energy consumption can be stated as follows:

$$Overall\ Energy\ Consumption = E_{task\_exe} + E_{sched\_overhead} + E_{static} + E_{shutdown} + E_{shutdown\_overhead} \quad (4.29)$$

$$E_{sched\_overhead} = T_{sched} \times E \quad (4.30)$$

where, T_{sched} is the total time required to perform scheduling events, E is the energy consumption per unit time by the processor core running at maximum frequency. T_{sched} can be calculated as follows:

$$T_{sched} = \mathrm{K1} \times \mathrm{p_{ac}} + \mathrm{K2} \times \mathrm{ap_{ac}} + \mathrm{L} \times \mathrm{cc} + \mathrm{M1} \times \mathrm{pc_{cold}} + \mathrm{M2} \times \mathrm{mc} + \mathrm{P} \times \mathrm{dc} + \mathrm{CST} \times (\mathrm{cc} + \mathrm{pc\_cold} + \mathrm{pc\_hot} + \mathrm{mc}) \quad (4.31)$$

where, $\mathrm{p_{ac}}$, $\mathrm{ap_{ac}}$, cc, $\mathrm{pc_{cold}}$, mc, dc and $\mathrm{pc_{hot}}$ are the number of periodic job arrival points, aperiodic job arrival points, completion points, preemption points with cold cache job, migration points, decision points and preemption points with hot cache job respectively. The time constants CST, K1, K2, L, M1, M2 and P are the time required for context switching, periodic job arrival, aperiodic job arrival, completion of a job, transfer data/instruction from L2 cache to L1 cache upon preemption, transfer data/instruction from L2 cache to L1 cache upon migration and the scheduler to make a scheduling decision respectively. These time constants and execution times are measured in milliseconds. The time constants are calculated by first running the scheduler code for large number of iterations and then taking average of all the iterations. These time constants are shown in table 3.4 of chapter 3. Preemption and migration cost of a job is considered as ten times the CST. It is assumed that the preemption and migration demand transfer of 3 and 5 pages respectively. Table 4.5 shows the energy consumption of the example tasks set with LAMCS, MCS and Non-DVFS algorithms over hyper period for WFD partitioning scheme.

Table 4.5: Energy Consumption over a Hyper Period (LAMCS)

Algorithm	**Execution Energy (m Joules)**	**Static Energy (m Joules)**	**Scheduler Energy (m Joules)**	**Shutdown Energy (m Joules)**	**Total Energy (m Joules)**
Proposed LAMCS	**67.63**	**0.187**	**3.23**	**0.967**	**79.888**
MCS	69.94	2.498	3.49	0.0	85.938
Non-DVFS	79.98	5.775	3.21	0.0	98.965
SVFS	73.65	4.505	3.21	0.0	91.359

4.9 Results and Discussions

This section provides details of the experiments conducted for the evaluation of the proposed scheduling algorithm, LAMCS. The experimental set up used for the experiments is same as in chapter 3 (section 3.8). LAMCS is compared with the scheduling algorithms namely, non energy aware (Non-DVFS), Static Voltage and Frequency Scaling (SVFS) and DVFS based Multi-Core Scheduler (MCS) scheduling algorithms. The performance is evaluated on the basis of two relevant metrics: overall energy consumption and response time of aperiodic tasks. In addition to these metrics, we have also identified the effect of our approach on various other parameters such as number of preemptions, migrations and scheduling decisions for two different partition techniques: WFD and FFD by varying the number of periodic tasks per task set, aperiodic tasks per task set, total periodic utilization and number of processing cores. The significance of all the parameters is discussed in chapter 3 (Section 3.7).

4.9.1 Performance Analysis of Proposed LAMCS considering only Periodic Tasks

Figures 4.3 and 4.4 compare the performance of LAMCS with the existing algorithms that schedule only periodic task sets. In this case, LAMCS is applied on periodic task set instead of mixed task set and the rest of the analysis that is shown in figures 4.5 to 4.21 is performed for mixed task sets consisting of periodic and aperiodic tasks.

In figure 4.3 and 4.4, the existing scheduling algorithms used for comparison are non energy aware (Non-DVFS) (Digalwar et al., 2014), cycle conserving scheduling algorithm (Pillai and Shin 2001) and Leakage Control Earliest Deadline First (LC-EDF) (Lee et al., 2003). These algorithms schedule only periodic task sets.

Figure 4.3 shows that LAMCS gives nearly equal energy consumption as Cycle Conserving algorithm and performs better than non-DVFS and LC- EDF. The reason for nearly equal performance of LAMCS and cycle conserving is that the probability of finding the shutdown intervals that are larger than the threshold time interval is less as the load on each core is balanced. Thus LAMCS performs nearly equal to the cycle conserving algorithm.

On the other hand, in figure 4.4, as the partition scheme is FFD, the load is concentrated among the subset of the cores keeping the remaining cores in idle state; LAMCS saves significant energy as compared to all other algorithms. This is because; the idle cores are kept in shutdown state resulting in significant amount of static energy saving. Another observation shows that LC-EDF performs better than Cycle Conserving algorithm for low and moderate task utilizations as the number of idle cores (which consume static energy) are more when total task set utilization is low. For the periodic utilization values of 70 and 80, Cycle Conserving algorithm saves more energy than LC-EDF. This shows that, dynamic shutdown strategy is more effective for lightly loaded processors as compared to heavily loaded processors.

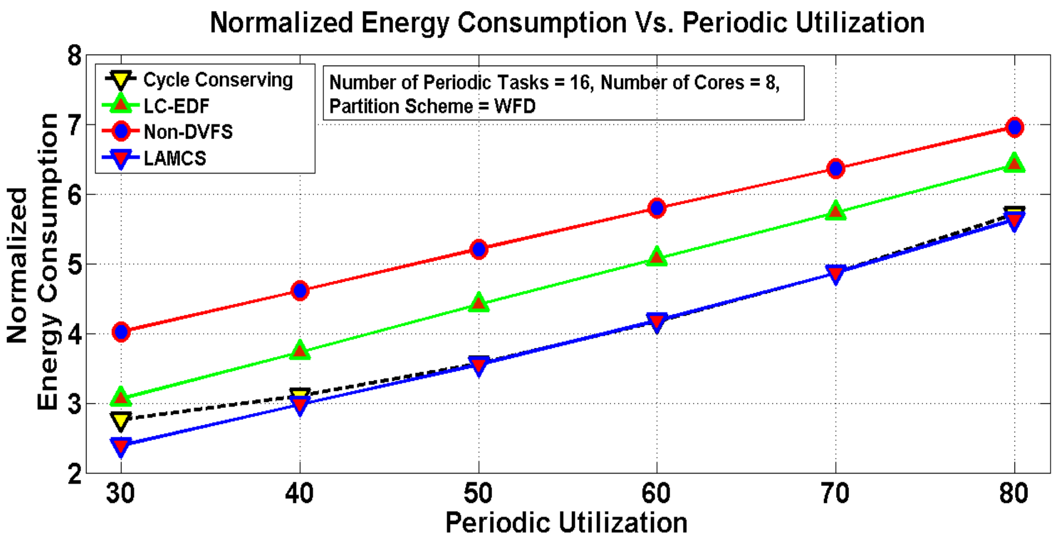


Figure 4.3: Normalized Energy Consumption Vs. Periodic Utilization (WFD partition scheme and task set constitutes only periodic tasks)

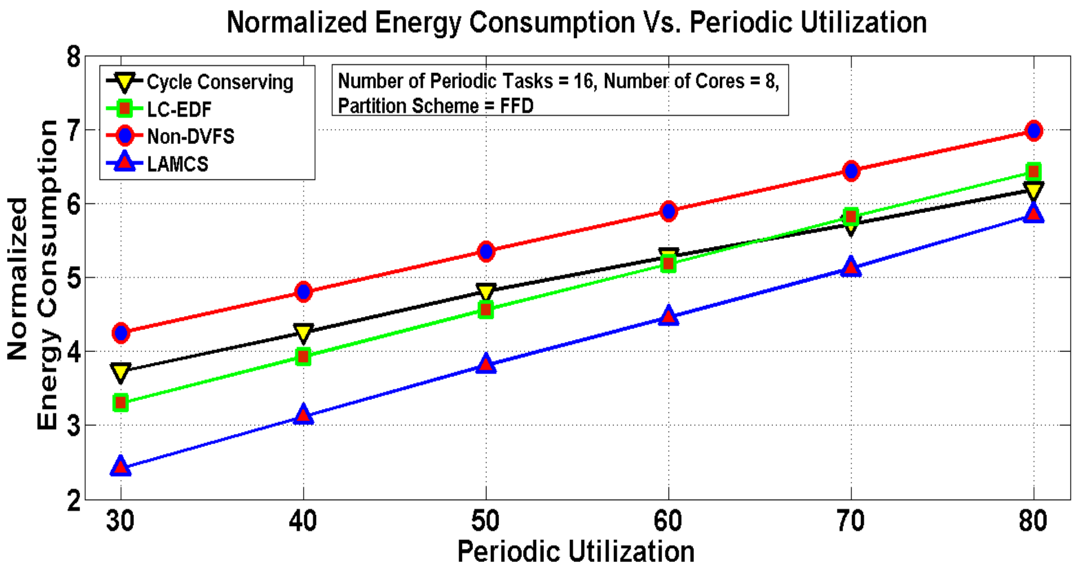


Figure 4.4: Normalized Energy Consumption Vs. Periodic Utilization (FFD Partition scheme and task set constitutes only periodic tasks)

4.9.2 Performance Analysis of Proposed LAMCS considering Mixed Task sets

The graphs shown in figures 4.5 to 4.19 show the effect on energy consumption, response time and scheduling events for mixed task sets.

4.9.2.1 Effect on Energy Consumption

Figures 4.5 and 4.6 show average normalized overall energy consumption for the mixed task sets with various algorithms against increasing periodic tasks while keeping number of aperiodic tasks, periodic and aperiodic utilizations and number of cores fixed. Figure 4.5 and 4.6 show the results for WFD and FFD partitioning schemes respectively.

The graphs in figures 4.5 and 4.6 show that the normalized energy measured with respect to LAMCS is less as compared to the other algorithms. Another observation is that MCS and LAMCS perform almost in similar fashion when partitioned with WFD where as LAMCS saves more energy as compared to MCS in case of FFD. This is because, the total shutdown period achieved in case of WFD is comparatively lesser than in FFD scheme and the number of shutdown intervals in case of WFD scheme are more as compared to FFD scheme. This leads to increase in energy consumed due to shutdown overhead. As a result, LAMCS using WFD scheme does not show significant reduction in overall energy as compared to MCS. On the other hand, in case of FFD partitioning scheme, the effect of shutdown is significant such that the overall energy consumption with respect to LAMCS is quit less as compared to MCS and other algorithms. In this case, the cores to which, not a single periodic task is assigned during task partitioning are completely shutdown. As a result, there is significant overall energy saving. In both the partitioning schemes, the energy consumption increases with increasing number of periodic tasks due to scheduling overhead of increasing number of tasks.

The graph in figure 4.7 shows the normalized energy consumption by varying total worst case utilization of periodic tasks in the mixed task set by keeping number of cores, number of periodic and aperiodic tasks fixed for FFD partitioning scheme. The overall energy consumption increases as the worst case periodic utilization is increased. This is due to increase in total execution time of the task set. LAMCS performs better than the other algorithms as it reduces both static and dynamic energy.

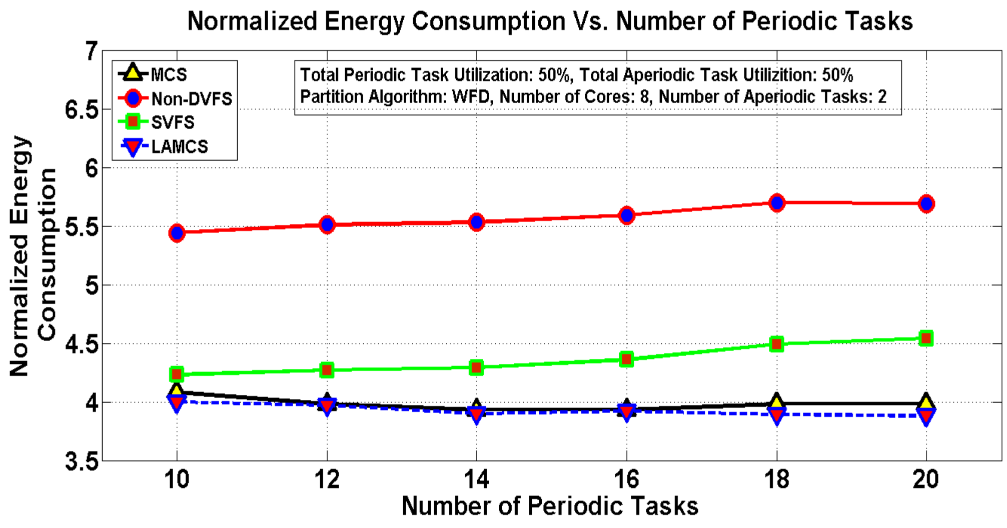


Figure 4.5: Normalized Energy Consumption Vs. Number of Periodic Tasks (WFD Partition Scheme)

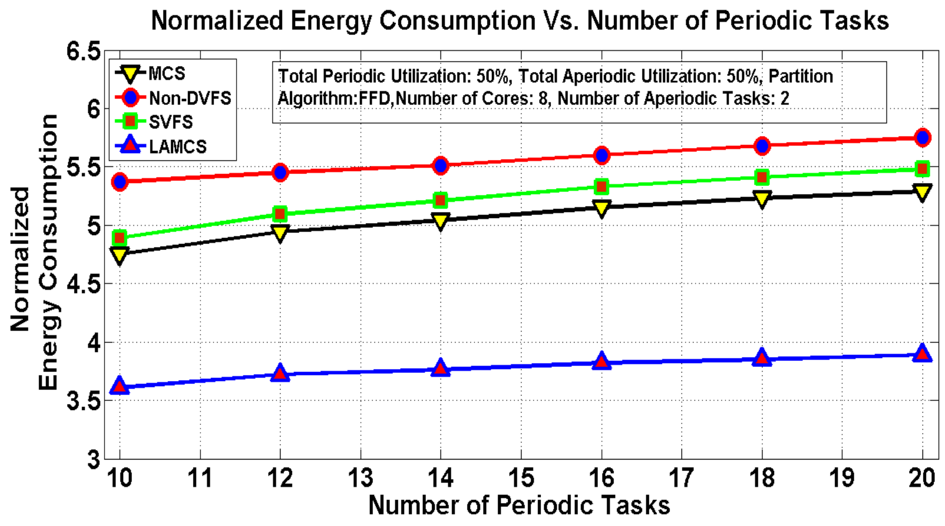


Figure 4.6: Normalized Energy Consumption Vs. Number of Periodic Tasks (FFD Partition Scheme)

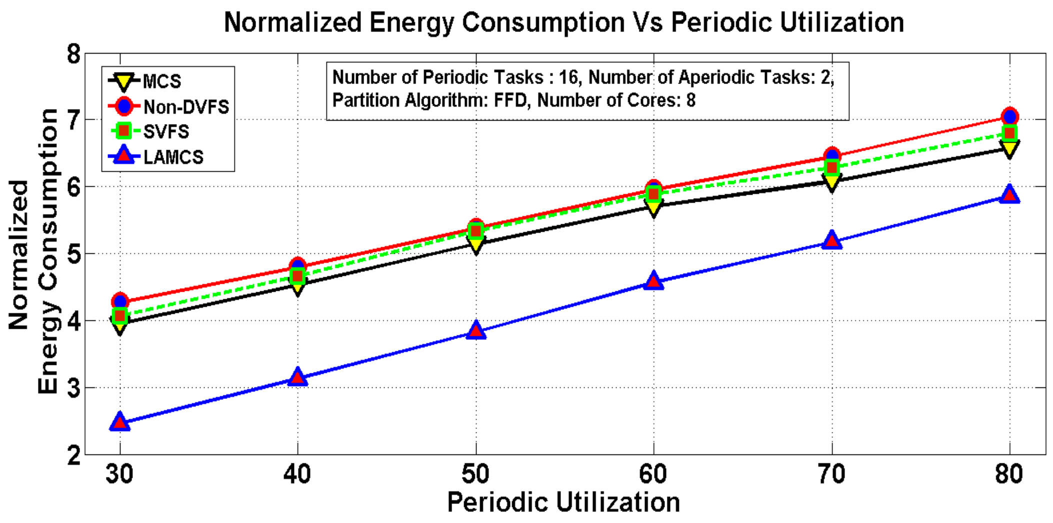


Figure 4.7: Normalized Energy Consumption Vs. Periodic Utilization (FFD Partition Scheme)

Figure 4.8 shows the effect of increasing aperiodic tasks on the overall energy consumption for FFD partitioning scheme. The parameters that are fixed to perform this experiment are total worst case periodic utilization, aperiodic utilization, number of processing cores and number of periodic tasks. The fixed values are shown in the graph. There is no increase in energy consumption with the increase in number of aperiodic tasks. This is because of fixed utilization of aperiodic tasks.

We have also investigated our algorithm by increasing the number of processing cores. In figures 4.9 and 4.10, normalized energy consumption is shown with respect to increase in the number of cores for WFD and FFD schemes respectively. The periodic utilization, number of periodic tasks and number of aperiodic tasks are fixed. The periodic utilization is fixed to 50% with respect to four cores.

In case of WFD scheme in figure 4.9, increasing the number of cores does not affect the execution energy (or dynamic energy) consumption but increases the static energy consumption due to increase in idle time. In addition, the inherent energy that is consumed to keep the processor in switch-on state also increases as the number of cores increase. Therefore, irrespective of the algorithms on the graph, the overall energy consumption increases as the number of cores increase. Among all the algorithms, LAMCS performs better than the others. This is because; LAMCS gets more opportunity to shut the cores as the number of cores increase.

In figure 4.10, energy consumption measured with respect to LAMCS is constant because FFD partitioning will remain unchanged even if we increase the number of cores. Therefore, even if we increase the cores, they will remain in shutdown state as no tasks will be assigned to them. For other algorithms, the scaled frequency selected for each core will be closed to maximum as FFD distributes the load in such a way that it tries to pack each core and then moves to the next core. They also show a slight increasing trend due to increase in static energy caused due to increase in idle time.

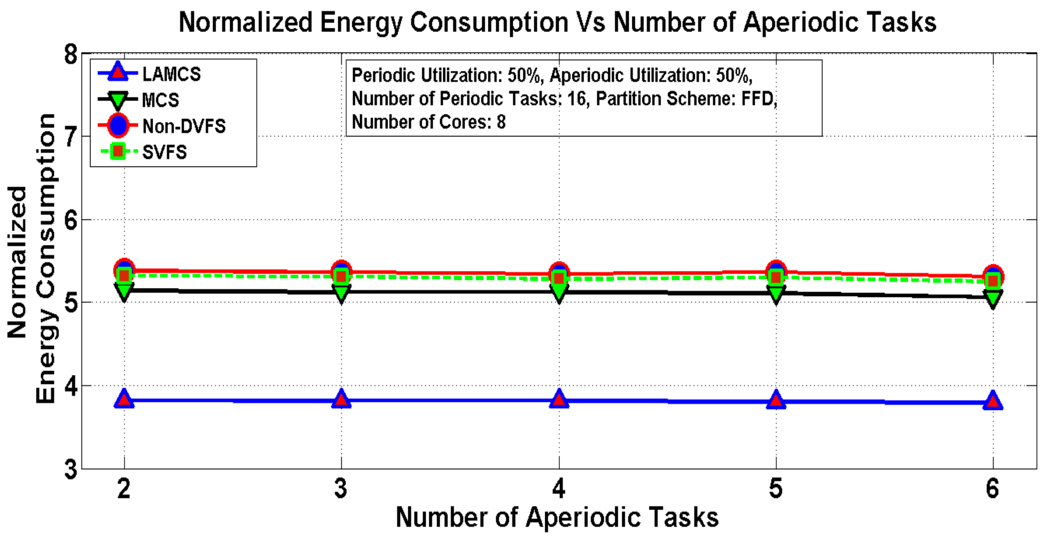


Figure 4.8: Normalized Energy Consumption Vs. Number of Aperiodic Tasks (FFD Partition Scheme)

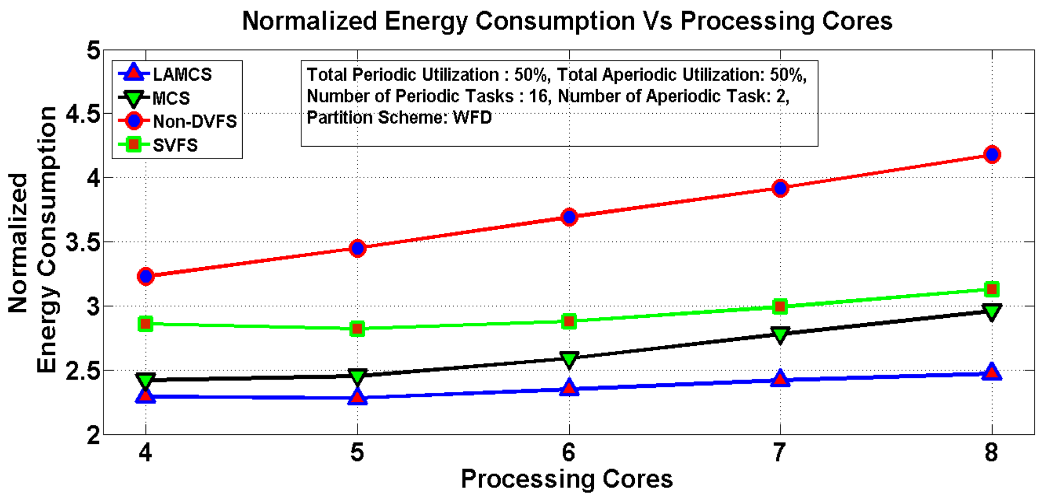


Figure 4.9: Normalized Energy Consumption Vs. Processing Cores (WFD Partition Scheme)

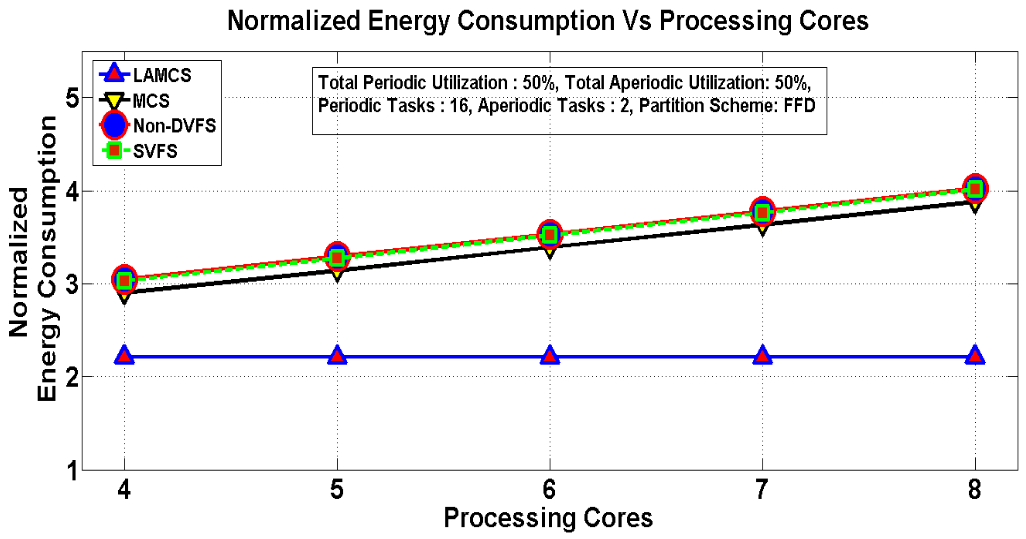


Figure 4.10: Normalized Energy Consumption Vs. Processing Cores (FFD Partition Scheme)

4.9.2.2 Effect on Response Time

Figures 4.11 - 4.15 show the effect of various parameters on aperiodic response time. In Figure 4.11, aperiodic response time measured by increasing periodic tasks for WFD scheme is shown. It can be noted that the response time increases by increasing number of periodic tasks. This is because; there is rise in scheduling events such as task arrivals, completions and preemptions which result in longer response time. In addition to this, LAMCS takes longer response time than other algorithms due to procrastination of aperiodic tasks that are arriving during shutdown period.

Figure 4.12 shows the aperiodic response time in case of FFD scheme. The algorithms other than LAMCS gives minimum possible response as these algorithms schedule the aperiodic tasks on lightly loaded core which does not have any periodic tasks assigned to it. But, in case of LAMCS, the cores that are not assigned the periodic tasks are kept in shutdown state. Therefore, the aperiodic tasks are executed on available cores with little higher response time. From figures 4.11 and 4.12, it is observed that the percentage increase in aperiodic response time by LAMCS with respect to other algorithms is higher in case of FFD than in WFD scheme. Figures 4.13 and 4.15 shows response time with respect to other two parameters: number of aperiodic tasks and number of cores for FFD scheme. These figures also show the same trend as seen in Figure 4.12 for the same reason as justified for Figure 4.12.

Figure 4.14 shows the affect of aperiodic response time on increasing number of processing cores. It can be noted from the graph that as the number of processing cores increase, the response time decreases because of decrease in current load on each core. The algorithms, MCS and LAMCS calculate virtual deadline based on the current processor utilization which is calculated dynamically and not on the basis of worst case processor utilization. Therefore, these algorithms are able to identify lightly loaded cores more accurately than Non-DVFS and SVFS algorithms resulting in decrease in response time. On the other hand, as Non-DVFS and SVFS algorithms calculate virtual deadline on the basis of worst case utilization of the cores, they are not capable of identifying the exact lightly loaded cores. In case of LAMCS, with the increase in processing cores, the response time is more with respect to other algorithms. For example, when the number of cores becomes 6, 7 and 8, the response time measured by LAMCS is more than other algorithms. This happens because of increase in shutdown period with the increase in processing cores. Increase in shutdown period increases response time as the aperiodic job execution is procrastinated during shutdown period.

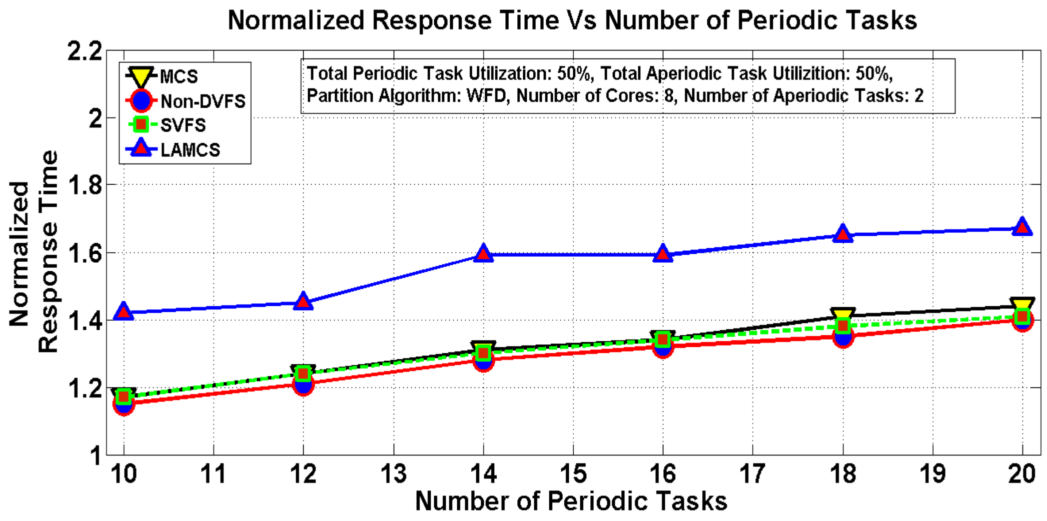


Figure 4.11: Normalized Response Time Vs. Number of Periodic Tasks (WFD Partition Scheme)

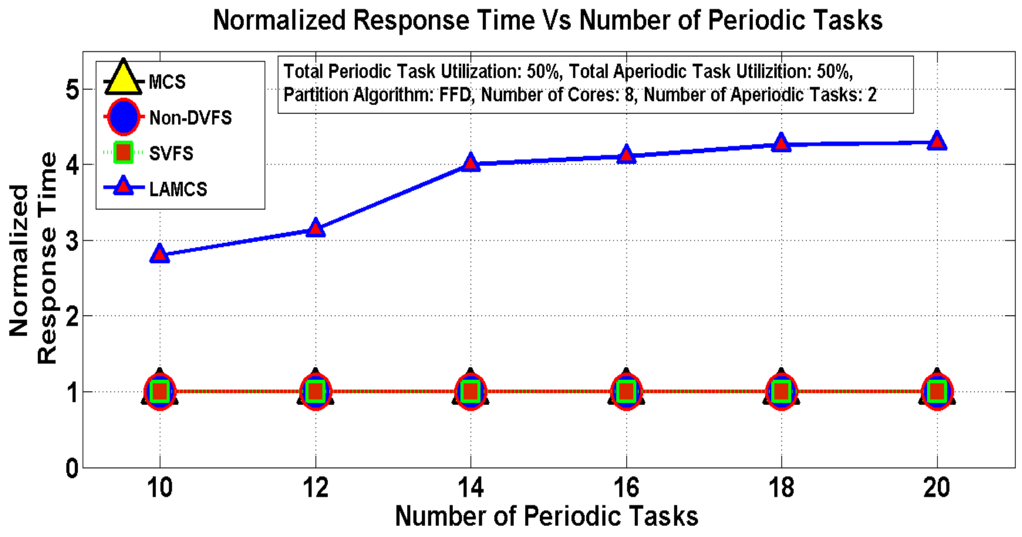


Figure 4.12: Normalized Response Time Vs. Number of Periodic Tasks (FFD Partition Scheme)

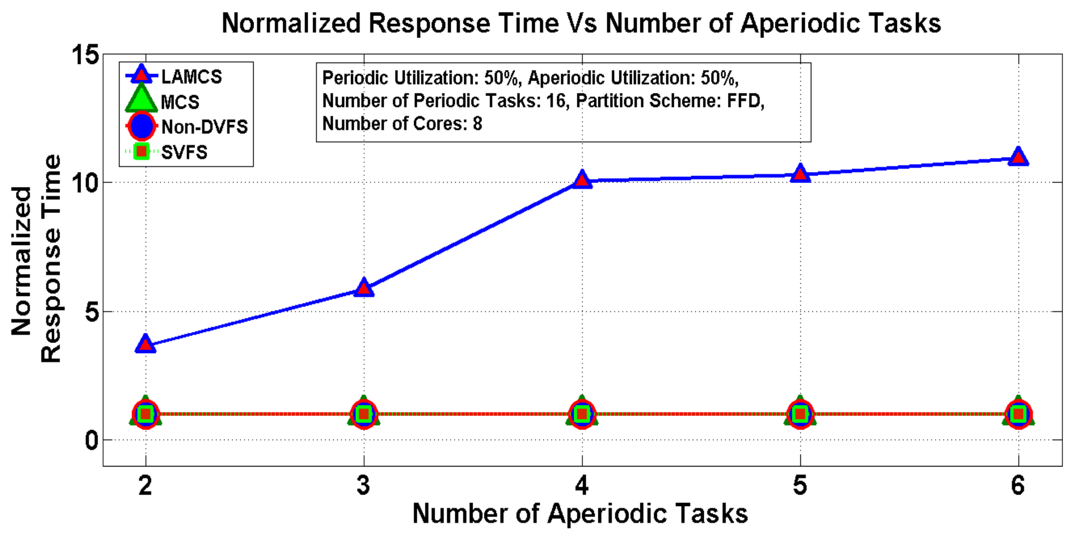


Figure 4.13: Normalized Response Time Vs. Number of Aperiodic Tasks (FFD Partition Scheme)

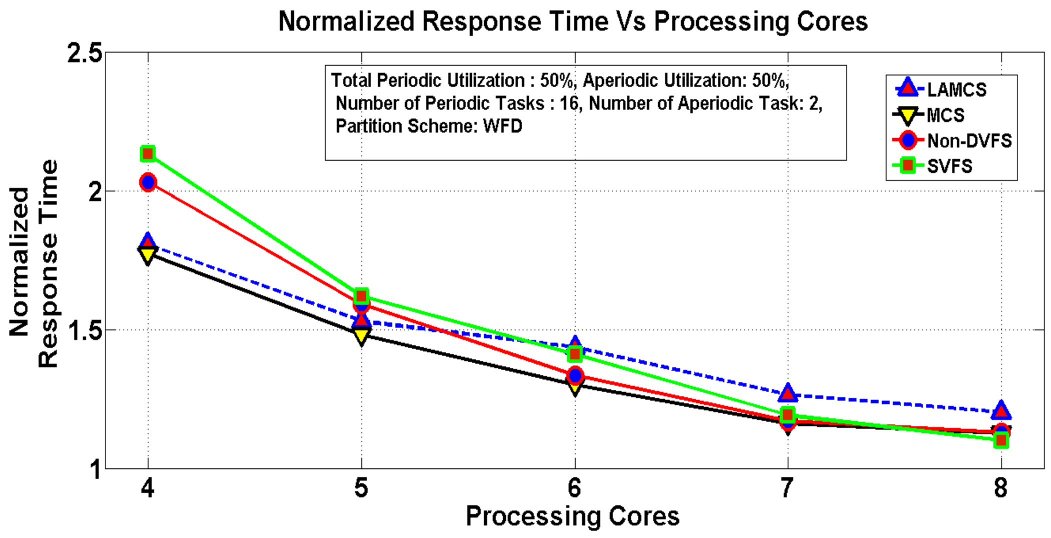


Figure 4.14: Normalized Response Time Vs. Processing Cores (WFD Partition Scheme)

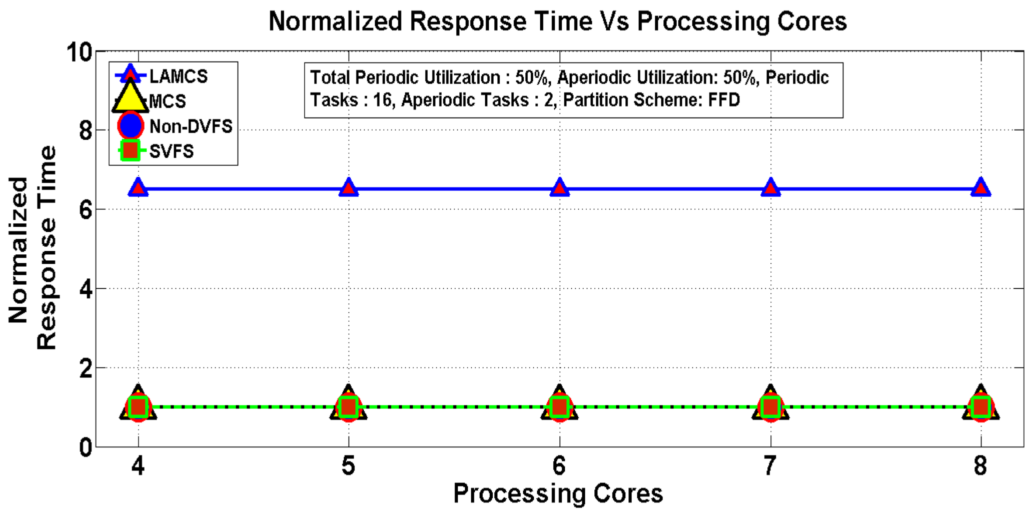


Figure 4.15: Normalized Response Time Vs. Processing Cores (FFD Partition Scheme)

4.9.2.3 Effect on Scheduling Events

The analysis of scheduling overhead in terms of number of preemptions, migrations and scheduling decisions is performed and the results are shown in the following graphs. The graphs in figures 4.16 and 4.17 show the variations in number of preemptions and scheduling decisions due to increase in number of periodic tasks for FFD scheme. The preemption count and scheduling decision points increase with increase in number of periodic tasks. This is because, by increasing the number of tasks in a task set, the probability of arrival of higher priority jobs increases resulting in increase in number of preemptions and scheduling decision points.

LAMCS has more number of preemptions and scheduling decision points because of two reasons: (1) It takes more execution time as compared to other algorithms due to frequency scaling. (2) The procrastination of tasks during shutdown period accumulates more number of jobs in the ready queue at the wake up time which results in more number of preemptions and scheduling decision points.

The graphs in figures 4.18 and 4.19 show the effect of migration of aperiodic jobs with increase in periodic tasks and increase in aperiodic tasks respectively. In both the graphs, LAMCS requires few migrations but other algorithms do not require any migration as the lightly loaded cores that do not possess any periodic tasks are available for use. But in case of LAMCS, such cores are put in shutdown mode for energy optimization. Therefore, aperiodic jobs are scheduled on the cores that are open for use.

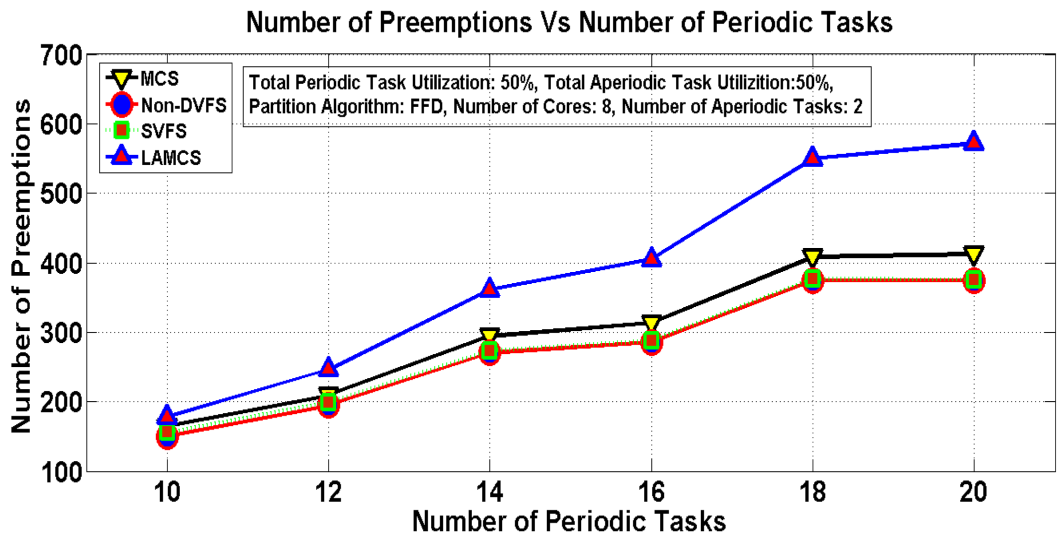


Figure 4.16: Number of Preemptions Vs. Number of Periodic Tasks (Partition Scheme: FFD)

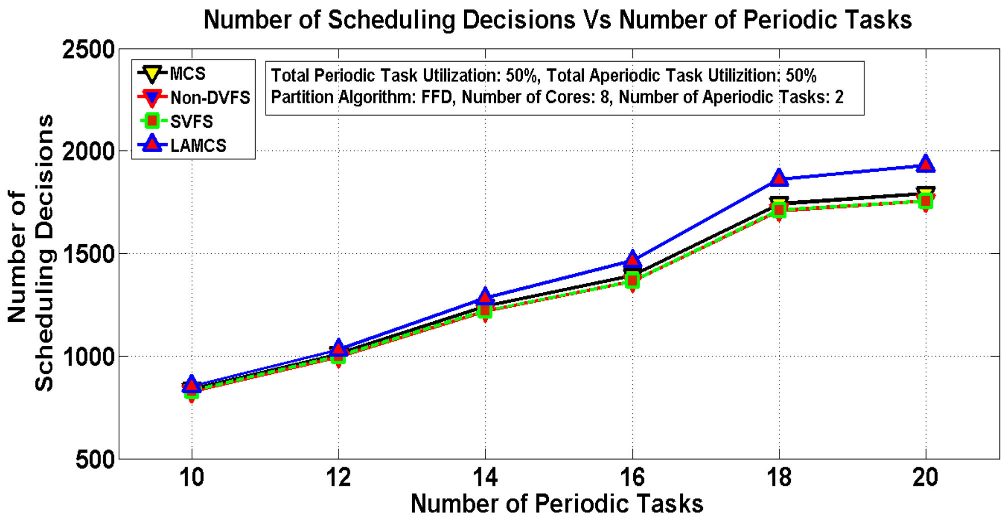


Figure 4.17: Number of Scheduling Decisions Vs. Number of Periodic Tasks (Partition Scheme: FFD)

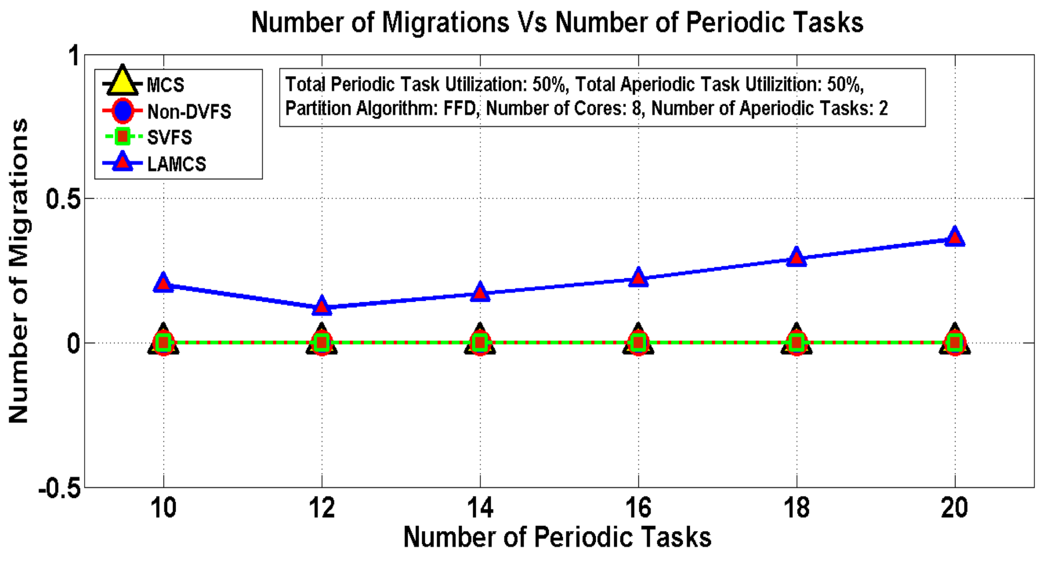


Figure 4.18: Number of Aperiodic Migrations Vs. Number of Periodic Tasks (Partition Scheme: FFD)

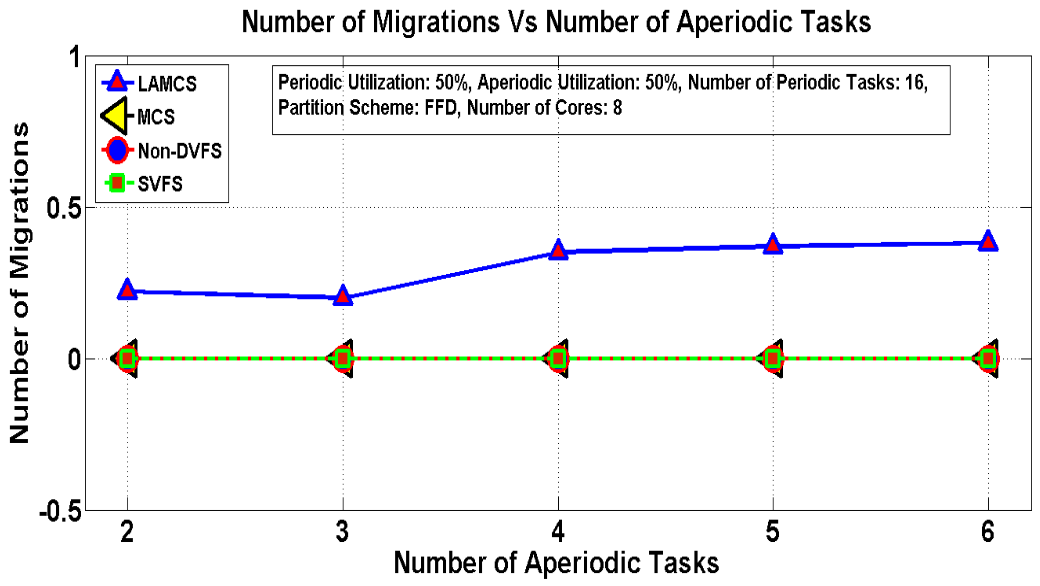


Figure 4.19: Number of Aperiodic Migrations Vs. Number of Aperiodic Tasks (Partition Scheme: FFD)

4.9.3 Comparison of Static Energy Consumption

In figures 4.20 and 4.21, the comparison of static energy consumption for both the proposed algorithms MCS and LAMCS is shown with respect to increasing periodic utilization for WFD and FFD partitioning schemes respectively. In LAMCS, as we have applied shutdown mechanism with procrastination of jobs, we consider the energy consumed during shutdown and shutdown overhead in addition to static energy consumption. Therefore, we compare the static energy consumption measured for MCS with sum of static energy, shutdown energy and shutdown overhead energy. It can be observed that LAMCS consumes quite less energy than MCS from both the graphs in figures 4.20 and 4.21. This major static energy saving contributes to the overall energy saving for LAMCS as compared to all the other algorithms.

In case of WFD scheme, the static energy consumption measured with respect to MCS and LAMCS decreases as the periodic utilization increases. This is due to increase in execution time which increases dynamic energy but not static energy. On the other hand, in case of FFD partitioning scheme, for MCS, static energy consumption decreases with increase in utilization. The reason is, as the total periodic utilization increases, the number of cores that are idle decrease. As a result, static energy consumption decreases. Instead, in LAMCS, as the utilization increases, the cores that are active also increase (or the number of cores that are in shutdown state reduces) resulting in slight increase in static energy.

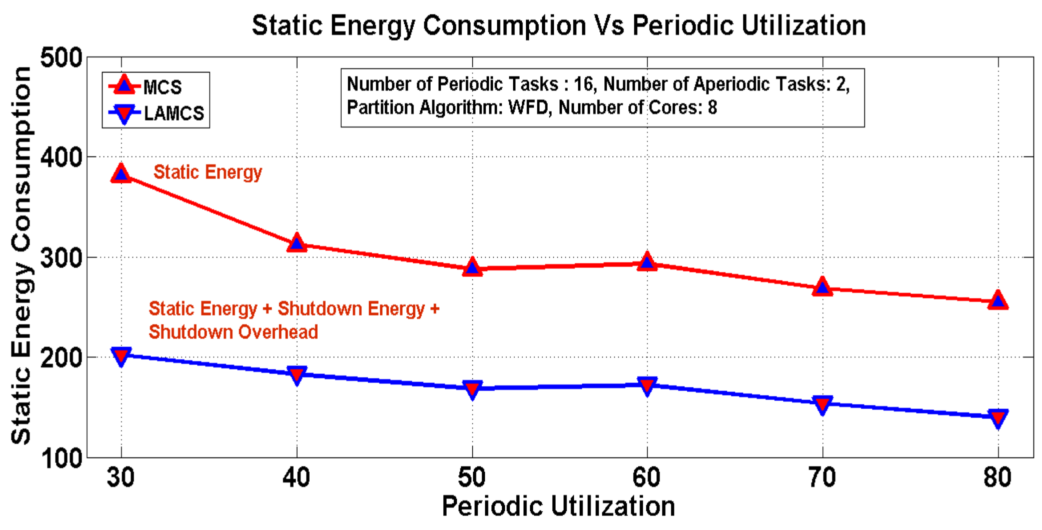


Figure 4.20: Static Energy Consumption Vs. Periodic Utilization (Partition Scheme: WFD)

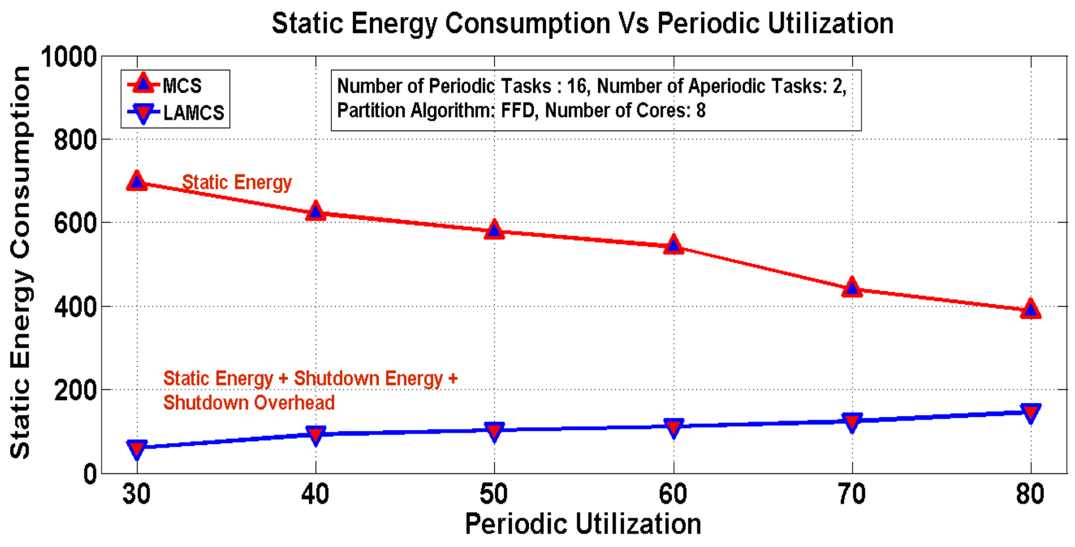


Figure 4.21: Static Energy Consumption Vs. Periodic Utilization (Partition Scheme: FFD)

It can be observed from the above analysis that LAMCS achieves more energy saving than the other algorithms with slight scheduling overhead and increased response time. Table 4.6 shows the percentage energy saving and percentage increase in average response time of aperiodic tasks measured using LAMCS algorithm with respect to other existing algorithms in case of FFD and WFD partitioning schemes.

Table 4.6: Performance of the Proposed LAMCS Algorithm

Partitioning Scheme	Comparison of LAMCS with	% Energy Saving using LAMCS	% Increase in Average Response Time using LAMCS
FFD	MCS	25.48%	73.4%
	Non-DVFS	32.1%	73.4%
	SVFS	27.89%	73.4%
WFD	MCS	1.51%	15.89%
	Non-DVFS	29.69%	15.64%
	SVFS	10.15%	16.23%

4.10 Summary

This chapter focused on the overall energy optimization that includes the optimization of dynamic and static energy. The energy optimization is performed by using real time scheduling algorithm for mixed task sets that contain periodic as well as aperiodic tasks on homogeneous multi-core processor. A leakage aware DVFS based real time scheduling algorithm, LAMCS, is proposed and implemented to optimize dynamic as well as static energy consumption of each processing core in a multi-core platform. The proposed algorithm LAMCS also optimizes the response time of aperiodic tasks by meeting the hard deadlines of the periodic tasks. LAMCS is sub-divided into two parts: (1) Task Allocation (2) Task Scheduling. Task allocation is done in similar way as in chapter 3. Task scheduling does scheduling of tasks (using EDF and TBS algorithms), takes care of optimizing dynamic energy consumption using DVFS based technique and static energy consumption using dynamic shutdown and procrastination techniques. It does not allow frequency scaling below critical speed. In order to reduce the shutdown overhead, it tries to extend the shutdown intervals using procrastination technique. Procrastination technique delays the execution of a task until the processor wakes up. Aperiodic tasks are handled in the same way as in MCS except that if the task arrives during the shutdown interval, its virtual deadline is calculated with respect to wake up time.

LAMCS is implemented using the proposed simulation tool STREAM and the analysis is done on various parameters such as energy consumption, aperiodic response time, preemption count, scheduling decision points, migration count etc. The behavior of the algorithm is tested on these parameters by varying the number of periodic tasks, number of aperiodic tasks, number of processing cores and by increasing total utilization of the periodic tasks in a task set. For each performance metric, 100 runs were made on 100 different randomly generated task sets.

On the basis of the simulation results, it is observed that, in case of FFD partitioning, the proposed algorithm LAMCS gives significant energy saving as compared to Non-DVFS, SVFS and MCS scheduling algorithms. It achieves 25.48%, 32.1% and 27.89% energy saving as compared to MCS, Non-DVFS and SVFS algorithms respectively. There is little overhead in terms of preemptions, migrations and other scheduling events that consume some energy. The aperiodic response time achieved by LAMCS in case of FFD partitioning scheme is more as compared to others. In case of WFD partitioning scheme, LAMCS consumes nearly equal amount of energy when compared with MCS resulting in less energy saving. In case of WFD partitioning scheme, LAMCS achieves 1.51%, 29.69% and 10.15% energy saving as compared to MCS, Non-DVFS and SVFS algorithms respectively. There is small percentage increase in aperiodic response time as compared to other algorithms. However, all the algorithms including LAMCS achieve nearly optimal response time with WFD partitioning scheme. The limitation of LAMCS is that in case of WFD partitioning scheme, the number of shutdown intervals are more and as a result, energy consumed in shutting down and waking up the processor core consumes significant amount of energy.

Chapter 5

Task Set Generator and Scheduler Simulator: STREAM

This chapter discusses the details of the task set generation algorithm used in this work and the design and implementation details of the scheduler simulator "STREAM" that is developed to test various energy aware and non-energy aware real time scheduling algorithms.

5.1 Introduction

Recent advancement in real time systems has led researchers to design and develop more efficient real time schedulers. To test the correctness and feasibility of these schedulers, it is necessary to examine them on a variety of real time task models and for huge number of task sets. Doing this manually is a cumbersome job and correctness is also not guaranteed. This necessitates the need of an automation tool that performs these jobs with greater flexibility and ease.

In order to evaluate the efficiency of a new real time scheduling algorithm, software simulation against other algorithms is commonly used. In real time system community, very few such tools are available and of the available tools, none is robust. These tools are mostly built for specific needs and do not cover all the aspects of real-time scheduling. STREAM is one such tool which is designed to be robust, flexible, and extensible. It serves as an automation tool for simulation, testing and evaluation of real time multiprocessor scheduling algorithms. In addition to existing features of existing simulators, STREAM particularly implements the missing aspects like adding robustness and flexibility, aperiodic task scheduling, power/energy management, performance analysis etc.

This chapter presents an easy to use and well documented simulation tool called STREAM that can simulate DVFS based real time scheduling algorithms for mixed work load on multi-core processor by incorporating majority of QoS parameters. STREAM stands for "Simulation Tool for Real time Energy efficient scheduling and Analysis for

Multi-core processors". It includes implementation of EDF, EDF with TBS, cycle conserving EDF with TBS algorithms for uniprocessor and multi-core processor platforms, DVFS based multi-core scheduler implementation for mixed work load, MCS and leakage aware DVFS based multi-core scheduler, LAMCS. It has modules to generate synthetic task set and to calculate various performance metrics such as energy consumption, aperiodic task's response times, various decision counts such as scheduling points, preemption count, migration count, cache impact points etc. The analytical results of the algorithms can be visualized by plotting different graphs.

STREAM is written in java programming language which makes use of object oriented paradigm. The modules are organized in such a way that highly specific or similar objects are grouped inside a single package. This helps new programmers to quickly track the required module by navigating the group hierarchy. The graphical User Interface (GUI) of simulator is very user friendly and is easy to explore and use. The use of abstract classes facilitates addition of new modules in the current version of simulator. The snap shots corresponding to various modules in STREAM are presented in Appendix A.

5.2 Background

Majority of the researchers in real time systems community evaluated the correctness of new algorithms by using simulation on randomly generated task sets. As a result, many simulation tools have been developed in last few years. The detail summary of over 20 simulators is shown in Table 5.1. The summary of the existing simulators is made on the basis the parameters viz., task model, processor model, task set generation, energy optimization, performance analysis, programming language, scheduler profiling etc. It can be observed from Table 5.1 that there is not a single simulation tool that covers all the aspects of energy efficient real time scheduling. The proposed simulator, STREAM, deals with all the parameters.

Table 5.1: Summary of Existing Simulators

Sr. No.	Simulator	Task Model	Processor Platform	Language	Design	Performance Analysis	Scheduler Profiling	Task Set Generation	Energy	Open Source
1	**STRESS** (Audsley et al., 1994)	Periodic Task System	Uni-processor	Domain Specific Language	Pseudo code design	N	N	N	N	N
2	**Generic Simulator** (Vroey et al., 1996)	Periodic Task System	Uni-processor	C++	Complex, Non-Modular, Redundant , No flexibility to add new module	N	N	N	N	N
3	**GHOST** (Sensini et al., 1997)	Mixed Task System	Uni-processor	C	Simple, Modular design, Flexibility to add new module	N	Trace Generator	Trace Generator	N	N
4	**MAST** (Harbour et al., 2001)	Mixed Task System	Multiprocessor	ADA	Modular, flexible to add new component	Y	Y	N	N	Y
5	**RTSim** (Manacero et al., 2001)	Periodic Task System	Multiprocessor	C++	-	N	N	N	N	Y
6	**Java Simulator** (Jakovljevic et al., 2002)	Periodic Task System	Uni-processor	Java	Modular, Concurrent programming, Independent Component design	N	N	N	N	N
7	**YASA** (Blumenthal et al., 2003)	Periodic Task System	Multiprocessor	ANSI C	Modular and Flexible, supports RT-Linux and RTEMS	Y	N	N	N	N
8	**SimDVS** (Shin et al., 2003)	Periodic Task System	Multiprocessor	-	Modular design, Flexible	Energy Management	N	N	Inter-DVFS and Intra-DVFS Techniques	N
9	**Cheddar** (Singhoff et al., 2004)	Mixed Task System	Multiprocessor	ADA	Modular, Flexible, allows integration of third party components	N	N	N	N	N
10	**TORSCHE** (Sucha et al., 2006)	Periodic Task System	Multiprocessor	Matlab/ Simulink	Routine based design	Timing Analysis	N	N	N	Y
11	**Realtss** (Diaz et al., 2007)	Periodic Task System	Uni-processor	C/C++/TCL	Modular, flexibility to add new scheduler	N	N	N	N	Y

Sr. No.	Simulator	Task Model	Processor Platform	Language	Design	Performance Analysis	Scheduler Profiling	Task Set Generation	Energy	Open Source
12	**FORTAS** (Courbin et al., 2011)	Periodic Task System	Multiprocessor	Java	Modular, flexible, Follows OOPS paradigm	Comparison between different scheduler performance	N	UUnifast-Discard Algorithm ()	N	Y
13	**SPARTS** (Nikolic et al., 2011)	Periodic Task System	Multiprocessor	Java	Modular, flexible, Follows OOPS paradigm	Execution time Vs. Task Set size, Simulation time analysis	Scheduler overhead statistics	Y	Y	
14	**omNET** (Khalib et al., 2012)	Periodic Task System	Uni-processor	C++	Modular	CPU Utilization Vs. deadline size and deadline stolerance	N	N	N	Y
15	**Yartiss** (Chandarli et al., 2012)	Mixed Task System	Multiprocessor	Java	OOPS design, Modular, Flexible, Extensible, Reusability	N	Tracking Preemption points	UUnifast-Discard Algorithm ()	N	Y
16	**RTMultiSim** (Hangan et al., 2012)	Periodic Task System	Multiprocessor	-	Follows STORM like design	CPU utilization, parallelism degree	N	UUnifast-Discard Algorithm ()	N	N
17	**RealtssMP** (Ramrez et al., 2012)	Periodic Task System	Multiprocessor	C/C++/TCL	Modular, flexibility to integrate new scheduler	Y	Tracking Preemption points, migration points, Deadline miss points	N	N	Y
18	**ERTSim** (Pillai and Isha, 2013)	Mixed Task System	Uni-processor	C/C++	Modular and Structural paradigm of C/C++	Utilization upper bound test, Response Time Analysis, Processor Demand Analysis	N	UUnifast, UScaling and Ufitting	N	N
19	**GEN4MAST** (Rivas et al., 2014)	Periodic Task System	Multiprocessor	Python	Modular, flexible	CPU Utilization Analysis	N	UUnifast-Discard Algorithm ()	N	Y
19	**GEN4MAST** (Rivas et al., 2014)	Periodic Task System	Multiprocessor	Python	Modular, flexible	CPU Utilization Analysis	N	UUnifast-Discard Algorithm ()	N	Y
20	**STORM** (Urunuela et al., 2010)	Mixed Task System	Multiprocessor	Java	Generic and simple OOPS design, Modular, Flexible, Extensible, Reusability	N	Gives only statistical information about scheduler	N	DVFS Technique	Y

5.3 Architecture of STREAM

The architecture of STREAM clearly separates the hardware specification from software entities. Modular and flexible organization of design components makes it clearly understandable for any new programmer. The provision of abstract classes and interfaces makes the modules easily extensible to new implementation.

5.3.1 General Purpose Design

The generic design structure of any real time simulator comprises of input/output subsystem separated by the simulation core components. Input subsystem deals with providing input data to be simulated by simulator components such as schedulers, servers, controllers etc. Output subsystem produces the set of output as a result of simulation. The core simulation components comprise of simulation building blocks, software components, hardware entities, event manager etc. The architecture of STREAM follows the same standard design flow as shown in figure 5.1.

- ***Input subsystem:*** This subsystem provides the input configuration data to be simulated. This includes software configuration such as task set definitions, target hardware requirements, type of scheduler and some miscellaneous configurations such as power/energy specification (for energy efficient schedulers).
- ***Simulation Core:*** It provides the simulation core modules which are highly modularized and coupled together to carry out sound and flexible simulation procedures on input data. Simulation core interactively holds together the software and hardware systems. Software system provides a bunch of simulation basic building blocks such as real time task model, global data-structures, and active components such as schedulers, servers, energy controllers, event recorders, performance analyzer, profilers etc. Software components are executed on target hardware platform provided by hardware system, which includes multi-core processor, per-core job queues and energy/power controllers.
- ***Output Subsystem:*** This subsystem records and tracks the output produced by event recorder, performance analyzer, scheduler profiler and log trace generator. Also called as

result set, it is then used for visual analysis, scheduler profiling and examination, processor execution trace analysis etc.

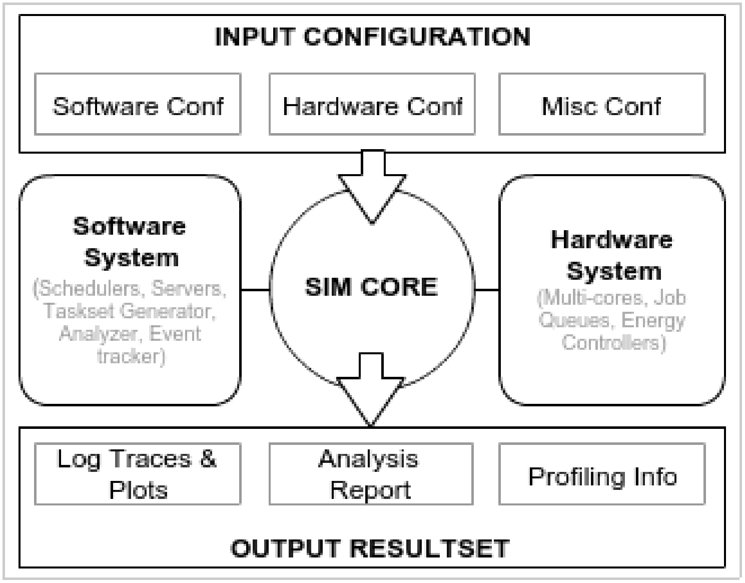


Figure 5.1: Abstract Model of STREAM

5.3.2 Subsystem Architecture

Subsystem architecture of STREAM provides the overall view of the system. It describes modeling and relationship between the different software and hardware entities in the system. These entities are categorized into four sub-systems:

- System Modeler
- Task set Generator
- Scheduler/Controllers
- Performance Analyzer/ Scheduling Profiler

The detailed architecture of the simulation tool STREAM is shown in figure 5.2. It shows the internal architecture of each of the sub-systems and describes the flow of control among the sub-systems.

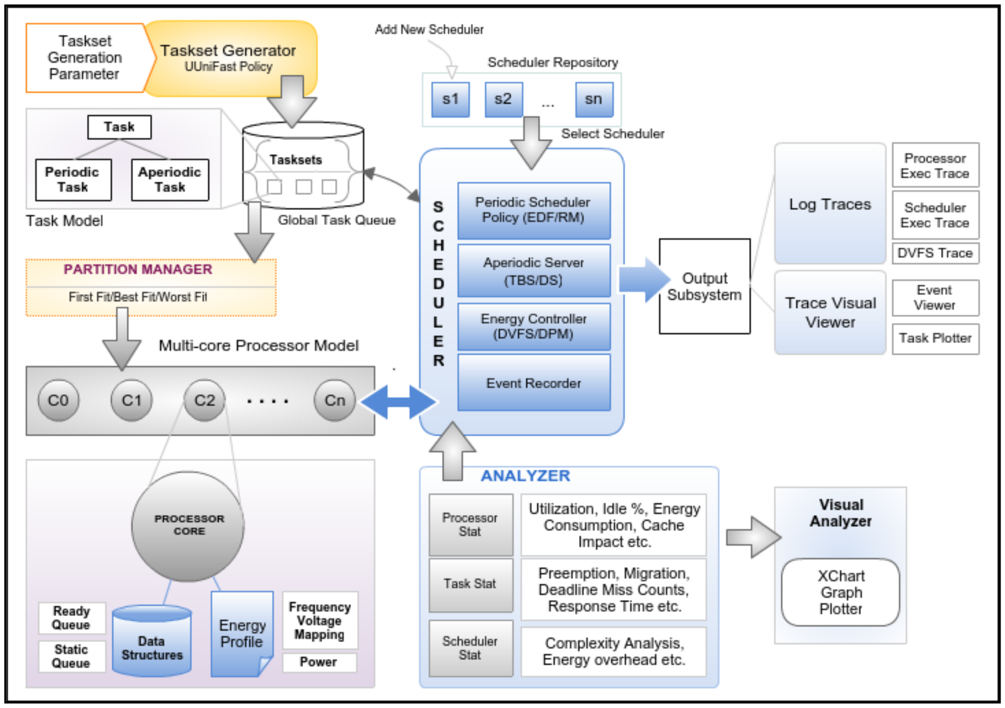


Figure 5.2: Architecture of STREAM

5.3.2.1. System Modeler

System modeler provides necessary working environment to facilitate and bring together the execution behavior of the simulator. It encapsulates *real time task entity*, *multi-core processor entity*, *energy model* and *partition manager*.

- ***Real Time Task Entity***

The *real time task entity* provides the characteristics of periodic and aperiodic real time tasks. These two types of tasks descend the task entity. Figure 5.3 shows the state transition diagram that shows all the events of a task throughout its life time. The events are explained as follows:

- The task begins with a state called *Zombie* where it lives in the static list of suspended tasks outside the execution environment.
- Upon its arrival, the task enters the *Ready* state where it is inserted into the ready queue of the system.

- Upon getting the chance, the scheduler makes it eligible for execution on the processor core, and the task enters *Running* state. This is depicted by a transition from *Ready* to *Running* state.
- While running, a task may get preempted by arrival of a high priority task in which case the scheduler brings back a task from its *Running* state to *Ready* state and gives chance to a high priority task.
- Later, on getting a chance, the preempted task again jumps to *Running* state.
- On termination, a task is moved back to *Zombie* state, where it waits until its next invocation (in case of Periodic task) or it marks its finished status (in case of aperiodic task).
- In addition to the basic states explained above, few scheduling algorithms may allow tasks to be migrated from one processor core to another (source execution unit to destination execution unit) on preemption. This helps reduce the waiting time of a task and gives lesser response time. This state is depicted in dotted lines (indicating that it's a scheduler specific implementation) and is called as *Migrated Ready*. On migration, it follows the same fundamental state behavior as explained above on the target processor.

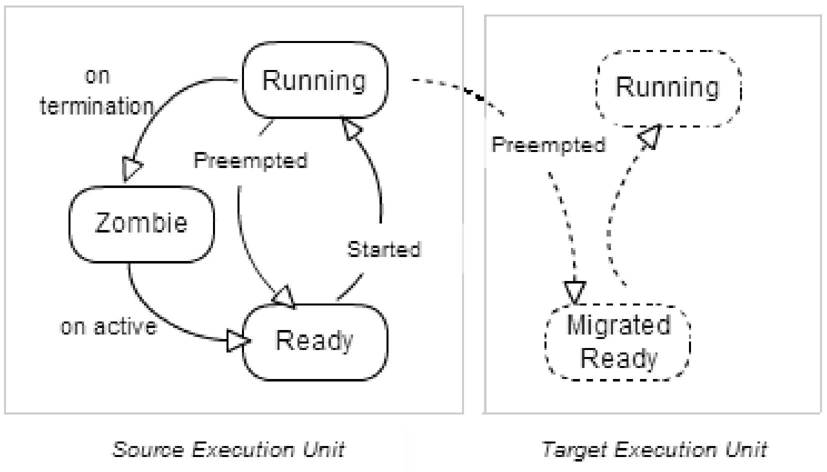


Figure 5.3: Task State Transition Diagram

- ***Multi-core Processor Entity***

The *multi-core processor entity* represents the virtual multi-core processor. The *Basic Processor* Entity provided in STREAM can be extended to implement a more specific processor entity by including the semantics of standard architecture of Intel, ARM etc. Multi-core environment is provided by creating multiple instances of the processor entity which can then be used to run the scheduler. As shown in figure 5.4*,* the processor entity consists of sub modules – *Task Data structure* and *Energy Profile.*

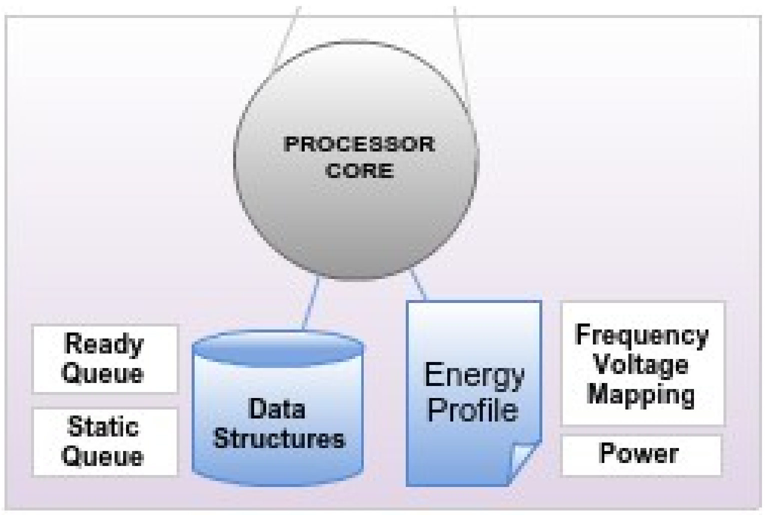


Figure 5.4: Processor Entity Diagram

- ***Task Data Structure***

The data structures that are designed for maintaining the task's information and energy consumption information of a processor core are described as follows:

- *Global Task Queue*: This maintains a global static list of tasks. A partition manager operates on this static list to perform partition across the processor cores according to partitioning scheme. This list is also useful in taking global scheduling decisions.
- *Per-Core Static Queue*: This queue is used to maintain a per-core static list of tasks. Partition manager partitions the tasks across the cores and places

them in per-core static-queue for further processing. Initially all the tasks are in *Zombie* state.

- *Per-Core Ready Queue*: Each core has its ready queue to keep track of all the ready tasks at a particular instance. This is basically a dynamic list of ready tasks managed by scheduler. Periodic tasks enters ready queue at every invocation and are removed after they finish their execution for one invocation. Aperiodic tasks on the other hand, enter ready queue only once at their arrival time and are removed after they are migrated or finish their execution.

- ***Energy Profiler***

  The energy specification of a processor depends on the type of architecture semantics used to build a processor model. For example, Intel's energy specifications are different from ARM's energy specification. The energy model in general represents various power or energy management techniques, frequency specification, and voltage specification that works on top of energy profile embedded inside the processor. In the current version of STREAM, DVFS and DPM with procrastination techniques of energy optimization are implemented.

- ***Partition Manager***

  In case of partitioned scheduling policy, tasks need to be partitioned across multiple cores. There are various partitioning schemes available in the literature. Few of these are: Worst Fit, First Fit, and Best Fit Partitioning Schemes (Gray and Johnson 1979). These schemes are implemented and encapsulated inside a partition manager module which is invoked prior to scheduling. There is a provision of adding new task allocation techniques if required in future.

5.3.2.2. Task set Generator

A rich set of task sets with uniform and even distribution serves as an important prerequisite to test the correctness and validity of any new scheduler. STREAM provides an independent module for generating synthetic task sets. The procedure of task set generation is very flexible in the sense that the user can set or modify the input task set parameters according to various requirements such as utilization, hyper period etc. These parameters are explained in detail below.

- ***Task Set Generation Parameters***

  There are two categories of task set generation parameters:

  - *Default Parameters*: These are the common parameters that are applicable for task set generation with their default values. These parameters are kept constant for generating large number of task sets. The default parameters are number of periodic tasks, number of aperiodic tasks, total utilization of periodic tasks, total utilization of aperiodic tasks and minimum number of target cores.
  - *User Parameters*: These are the variable parameters that are accepted from user based on the requirement. The user parameters are number of target task sets, hyper period range, actual execution time range factor and aperiodic task arrival range factor. These parameters vary according to the requirement of utilization, number of periodic tasks per task set, number of aperiodic tasks per task set, number of cores etc.
  - *Brief description of all the parameters:*
    - *Periodic tasks load (utilization)* represents the total utilization of periodic tasks present in one task set for single processor core. This utilization should be less than or equal to 1(<= 100% Load).
    - *Aperiodic tasks load (utilization)* represents the total utilization of aperiodic tasks over a span of one hyper period. For example, if for a task set hyper period is HP and *Aperiodic tasks load* is 70%, then the total aperiodic task load over a span of one hyper period = 70% * HP.
    - *Number of task sets* represents the total number of task sets of similar configurations to be generated. The default value is 100 which can be changed in default parameters settings of task set generator.
    - *Hyper period range* represents the range factor which is used to constrain the hyper period value from generating a very large number. The default range given in

STREAM is 360-3000; however, this can be changed in default parameter settings of task set generator.

- *Actual Execution Time (AET) generator* represents a boolean field to allow generating AET values over a hyper period corresponding to WCET value of a task.
- *AET range factor* is used to randomly generate AET values constrained within this range factor. To generate an AET value corresponding to a WCET value of a task, a random factor from within this range is selected and multiplied with a WCET. This guarantees, the AET values generated are less than or equal to the WCET value. The default range factor used is 0.30-0.95 of WCET. For ex, WCET = 20 and a randomly selected factor within this range is 0.5(say), then the AET = 0.5 * 20 = 10.
- *Aperiodic task arrival range factor* is used to decide the release time interval of aperiodic tasks. From a given range factor of 0.01-0.10(say), if 0.02 is the value selected randomly, then first aperiodic task will arrive at time = 0.02 * hyper period. The range factor for next aperiodic task will be upgraded to 0.11-0.20 and a randomly selected value within this range multiplied by hyper period decides its arrival time, and so on for all successive aperiodic tasks.

- ***Task set Generation Policy***

STREAM is capable of generating mixed task sets that contain periodic as well as aperiodic tasks. The algorithm designed and implemented for the generation of mixed task sets is derived from a well known algorithm: UUniFastDiscard (Davis and Burns, 2009) which is capable of generating task sets with only periodic tasks.

The generation of synthetic task sets should meet three key requirements: efficiency, parameter independence and lack of bias. Efficiency in terms of ability to generate large number of task sets for each task set parameter setting in the experiments. This is required to get statistically significant results. The parameter independence refers to the ability of the algorithm to generate task sets by varying subset of parameters and keeping other parameters constant. For example, different task sets can be generated by increasing periodic tasks in a task set for fixed periodic and aperiodic utilization. The distribution of task sets generated should be equivalent to selecting task sets at random from all possible task sets and then discarding those that do not match required parameter setting.

The number of tasks and total utilization of the task set are the two important parameters required to generate the tasks in a task set. The attributes of a periodic task are worst-case execution time (C), period (P) and deadline (D). We assume implicit deadline where deadline is equal to period. The attributes of aperiodic tasks are arrival time and worst-case execution time. The total utilization of a task set can be defined as in equation 5.1:

$$\sum_{i=1}^{n} U_i = u \tag{5.1}$$

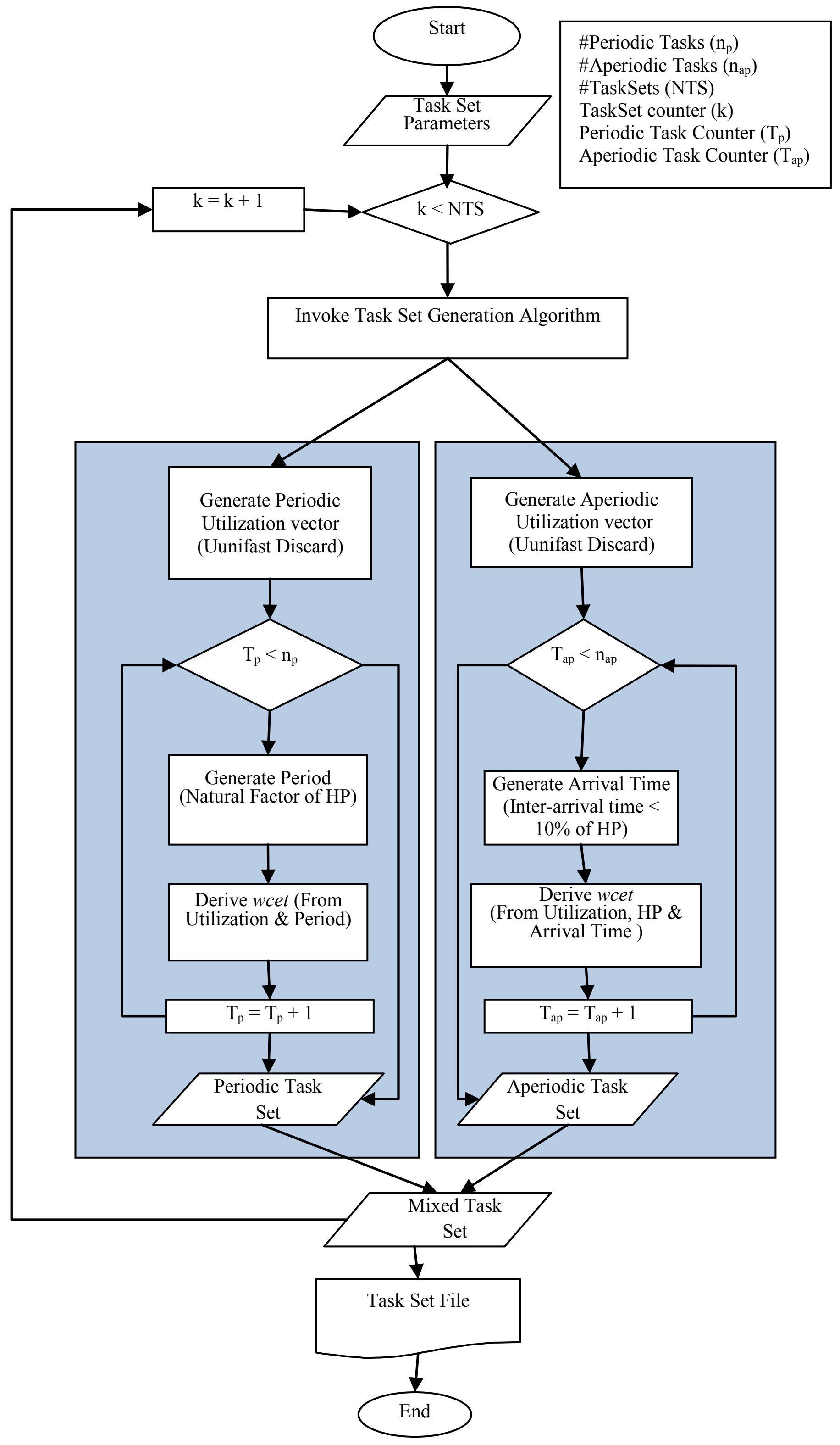
where,

$$U_i = \frac{C_i}{P_i} \tag{5.2}$$

where n is the number of periodic tasks, U_i is the utilization of ith periodic task, u is the total utilization of periodic tasks. For the generation of periodic tasks, utilization values for all the periodic tasks in a task set and corresponding periods should be randomly generated. The worst-case execution time can be computed from equation 5.2.

There exist various algorithms in literature for the generation of task utilization. In each of the algorithms, the task utilization is randomly generated with the constraint that the sum of utilizations should be constant desired total utilization of the task set. Bini and Buttazzo (2005) proposed two algorithms UUniform and UUniFast for uniprocessor platform where total utilization cannot exceed 1. UUniform algorithm is practically infeasible while UUniFast is an efficient algorithm. The logic behind UUniFast is to initially sample a value which represents the sum of n-1 task utilization values and then set a task utilization value to the difference between required total and this sampled value. This operation is repeated for each task for the sampled value in previous iterations as the required total. This algorithm works well for uni-processor platform but for multiprocessor platform, as the utilization of each task should not exceed 1, UUniFast cannot be directly applied. An algorithm UUniFast-Discard proposed by Davis and Burns (2009) is a simple extension to UUniFast algorithm which simply discards the tasks whose utilization is greater than 1. The limitation of this algorithm is that it becomes inefficient as total utilization value approaches n/2 where n is very large. Another efficient algorithm proposed by Stafford called Randfixedsum (Stafford, 2006) is efficient than the other algorithms. It efficiently generates a set of vectors that are evenly distributed in *n-1* dimensional space and whose components sum is equal to a constant value. The reason behind its efficiency is that the random samples are so generated that they are not required to be rejected.

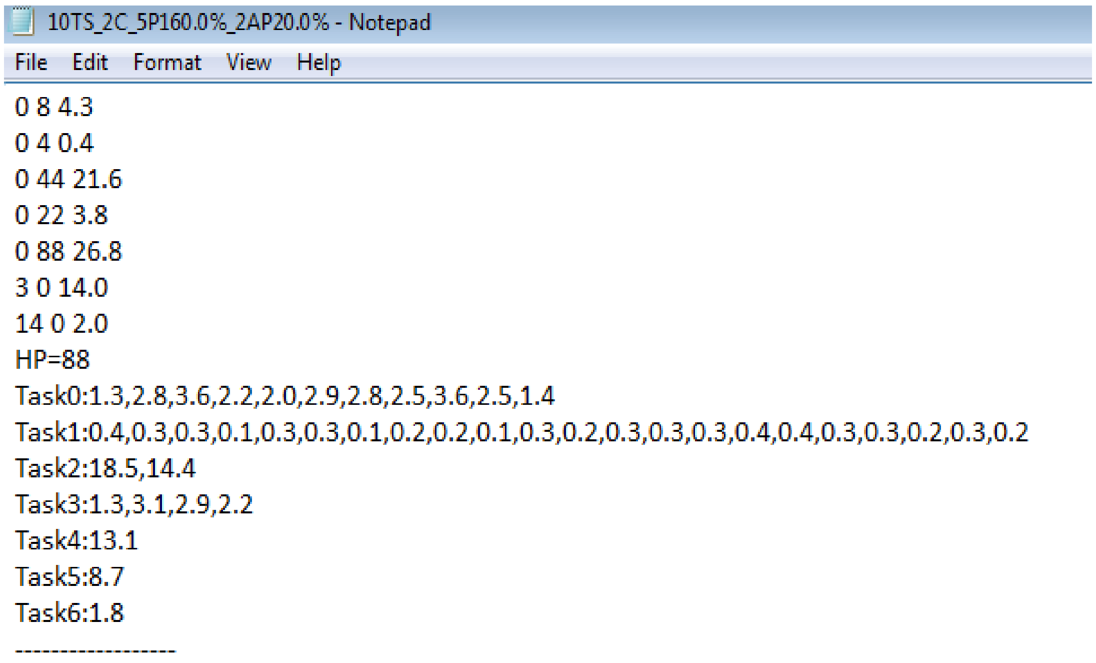
In our simulation tool STREAM, we have used UUniFast-Discard algorithm. Figure 5.5 shows the flowchart of synthetic task set generation with the attributes displayed in a separate box. The task set generation algorithm is responsible for generating periodic and aperiodic tasks. As shown in the flow chart, in each iteration, one mixed task set is generated. For generating periodic/aperiodic tasks, it makes use UUniFast-Discard (Davis and Burns, 2009). It generates a vector of uniformly distributed utilization values such that the sum of those utilization values is equal to periodic/aperiodic load. It also discards those utilization values that are exceeding 1 in case of periodic task. Period of each periodic task is chosen to be a random number such that it is a natural factor of a given hyper period value. Execution time is derived from the utilization value and period. Arrival time of the aperiodic task is obtained as a random number over a hyper period and the minimum inter-arrival time between two aperiodic tasks is not more than 5% to 10% of hyper period. The *wcet* of aperiodic task is calculated as a product of utilization value and the time left till the hyper period from its arrival. The synthetic task generator generates periodic tasks for a wide range of utilization: 30% to 80% and aperiodic tasks utilization is based on the remaining utilization of processor. The number of periodic tasks in a task set range from 2 to 20 and aperiodic tasks in a task set range from 2 to 6. For each class of fixed number of tasks and utilization, 100 task sets are generated. Task set generator is implemented as a generic interface that provides the flexibility of adding the new task set generation algorithm.



Start
Task Set Parameters
#Periodic Tasks (n_p)
#Aperiodic Tasks (n_{ap})
#TaskSets (NTS)
TaskSet counter (k)
Periodic Task Counter (T_p)
Aperiodic Task Counter (T_{ap})
k = k + 1
k < NTS
Invoke Task Set Generation Algorithm
Generate Periodic Utilization vector (Uunifast Discard)
Generate Aperiodic Utilization vector (Uunifast Discard)
$T_p < n_p$
$T_{ap} < n_{ap}$
Generate Period (Natural Factor of HP)
Generate Arrival Time (Inter-arrival time < 10% of HP)
Derive *wcet* (From Utilization & Period)
Derive *wcet* (From Utilization, HP & Arrival Time)
$T_p = T_p + 1$
$T_{ap} = T_{ap} + 1$
Periodic Task Set
Aperiodic Task Set
Mixed Task Set
Task Set File
End

Figure 5.5: Task Set Generation Flowchart

Figure 5.6 shows the sample task set file generated by the task set generator. Even though the current version of STREAM generates tasksets in .txt file, there is provision to add other useful file formats such as .csv, .xml etc. As shown in figure 5.6, two tasksets are separated by a dotted line (---) also called a separator. Each task set consist of 'p' number of periodic tasks and 'a' number of aperiodic tasks, where p > 0 and a >= 0. This is followed by the hyper period value for that task set and then the set of actual execution time (AET) values corresponding to each task.


10TS_2C_5P160.0%_2AP20.0% - Notepad
File Edit Format View Help

```
0 8 4.3
0 4 0.4
0 44 21.6
0 22 3.8
0 88 26.8
3 0 14.0
14 0 2.0
HP=88
Task0:1.3,2.8,3.6,2.2,2.0,2.9,2.8,2.5,3.6,2.5,1.4
Task1:0.4,0.3,0.3,0.1,0.3,0.3,0.1,0.2,0.2,0.1,0.3,0.2,0.3,0.3,0.3,0.4,0.4,0.3,0.3,0.2,0.3,0.2
Task2:18.5,14.4
Task3:1.3,3.1,2.9,2.2
Task4:13.1
Task5:8.7
Task6:1.8
------------------
```

Figure 5.6: A Sample Task Set File

As shown in figure 5.6, for each task set,

A periodic task is represented as:

ARRIVAL_TIME<space>PERIOD<space>WORST_CASE_EXECUTION_TIME

An Aperiodic task is represented as:

ARRIVAL_TIME<space>0<space>WORST_CASE_EXECUTION_TIME

Task set hyper period follows the task definition and is represented as:

HP=<HYPERPERIOD VALUE>

Actual Execution Time (AET) follows next, the format is

Task# :< RANDOM AET VALUES SEPARATED BY COMMA (,) FOR n (= HP / PERIOD) JOBS OF A TASK>

5.3.2.3. Schedulers, Controllers and Servers

The main objective of any real time simulator is to test, analyze and evaluate the scheduling algorithm by providing the necessary infrastructure around it. Simulator should provide the flexible and extensible environment to design new scheduler. In STREAM, Schedulers resides at the center of the simulator and are surrounded by the set of subsystems consisting of task entity, processor entity, analyzer, task set generator etc. STREAM provides the flexibility to add new scheduler to the simulator with minimum efforts.

- ***Scheduler Composition:***

In STREAM, a scheduler is composed of three executing components: Scheduler, Controller and a Server. Of these, scheduler is mandatory since it offers the basic scheduling policies such as EDF, RM etc. The other two are closely integrated with scheduler and are optional, meaning that they can be included depending on the type of scheduler we want to design. For example, an energy aware EDF scheduler would need to integrate DVFS controller inside basic EDF scheduler component. Following sections discuss in detail about these components.

o ***Scheduler:***

This is the fundamental component type necessary to be implemented in order to write new scheduler. Basic scheduling policies such as EDF and RM are implemented as a part of this component by implementing a Scheduler interface. This is the mandatory component for any scheduler and it provides the flexibility to integrate other two optional components.

o ***Controller:***

In order to make the basic scheduler energy aware, it has to be composed with energy controller(s). The implementation of energy controller depends on the type of power management technique to be followed. In the current version of STREAM, DVFS and shutdown with procrastination techniques of power management is followed which allows dynamic management of power at various task state events such as start, finished, preempted etc. Energy controllers relies on the energy specification provided by energy profile embedded inside the processor.

- ***Server:***

In order to look after the scheduling of aperiodic tasks, a special scheduler component is designated called as server or aperiodic server. The server can be embedded inside the basic scheduler in order to make it handle aperiodic tasks. STREAM provides flexible way of defining new aperiodic server. In current design, Total Bandwidth Server (TBS) is implemented for its simplicity and efficiency as compared to other servers such as Deferrable server, Sporadic server etc.

5.3.2.4. Current Design of STREAM

In the proposed simulation tool, various flavors of dynamic priority schedulers have been implemented. Though the current version of STREAM does not focus on fixed priority scheduling techniques, it provides the flexibility to add new schedulers. The schedulers currently implemented in STREAM are all based on EDF scheduling policy. The aperiodic server currently implemented is TBS and DS. To make these schedulers energy efficient, a dynamic energy optimization technique namely, DVFS is implemented. Also to optimize overall energy consumption, DVFS together with shutdown and procrastination techniques is implemented. The flexible design of simulator allows execution of different schedulers separately as well as in combination with different aperiodic servers and energy controllers depending upon the user requirement. For example, if a user wants to execute non energy aware scheduler for hard real time tasks, then EDF can be selected or if a user wants to execute energy aware scheduler for mixed task set, then a combination of EDF, TBS and DVFS can be selected. The scheduler, aperiodic server and energy controller are implemented as java interface that makes it easy to add new scheduling policy, another aperiodic server or any other energy optimization policy. Figure 5.7 shows the detailed class diagram of the simulation core which includes scheduler, server and energy controller.

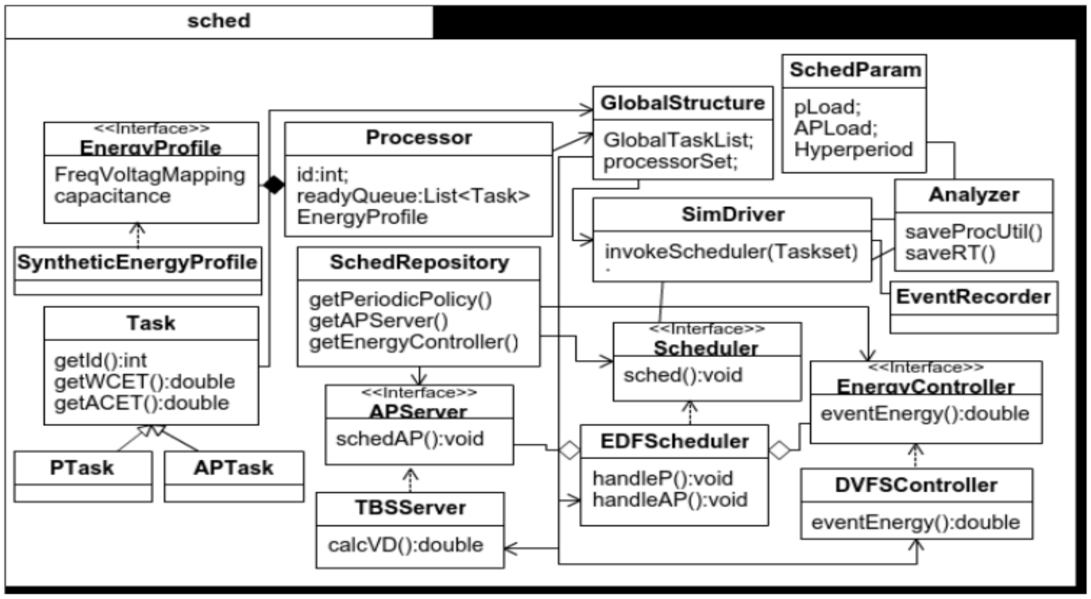


Figure 5.7: Class Diagram of Scheduler Module

The current version of simulator provides following building blocks that are necessary to design and build any advance real time schedulers.

- *Basic Scheduling Policies:*

A dynamic priority-scheduling algorithm, EDF is implemented which can generate a feasible schedule for a system of *N*-independent, pre-emptible tasks as long as the total utilization of the system is less than or equal to 1.

Every time an aperiodic task arrives, the TBS algorithm assigns the possible earliest deadline to the aperiodic task. Since the aperiodic task does not have a deadline, a virtual deadline is calculated for them. Once the task is assigned the virtual deadline, it is scheduled by EDF with periodic tasks. The virtual deadline is determined based on the server bandwidth (utilization).

- *Controllers*
  - ➢ *DVFS Controller:* Energy aware schedulers invoke DVFS controller in conjunction with EDF in order to perform power/energy management action. It is invoked at every decision event and DVFS actions such as frequency-voltage resetting, scaling of execution time etc. are performed.

- *Static Frequency Controller:* It is basically a static version of DVFS Controller, meaning that a dynamic frequency-voltage resetting doesn't happen and instead worst-case frequency-voltage settings are used throughout the schedule.
- *Leakage Aware DVFS Controller:* The algorithms that optimize the dynamic as well as static energy invoke DVFS controller in conjunction with a controller responsible for shutdown and procrastination. These controllers are invoked on top of basic EDF and TBS schedulers for mixed task set.

The current version of simulator includes the following real time scheduling algorithms designed at the top of above building blocks.

1. *Non-DVFS EDF+TBS Scheduler (Non-DVFS):*

This is non-energy aware version of EDF+TBS scheduler which doesn't apply DVFS for energy reduction and executes tasks at full frequency and consumes maximum processor power. The scheduler can work for uniprocessor and multi- core platforms.

2. *Static Frequency EDF+TBS Scheduler (SVFS):*

   This scheduler is capable of scheduling mixed task set on multi-core processors using EDF and TBS. It is able to optimize dynamic energy by scaling the frequency based on the worst-case utilization of the task set. It can work for uniprocessor as well as multi-core platforms.

3. *Cycle Conserving DVFS Enabled EDF+TBS Multi-core Scheduler (CC_MCS):*

   This scheduling algorithm is DVFS enabled where frequency scaling is based on cycle conserving approach followed by Pillai and Shin (Pillai and Shin, 2001).

4. *DVFS Enabled EDF+TBS Multi-core Scheduler (MCS):*

   This scheduling algorithm is also capable of reducing dynamic energy. The frequency scaling is done on the basis of dynamic utilization which is calculated at each scheduling point.

5. *Leakage Aware DVFS Enabled EDF+TBS Multi-core Scheduler (LAMCS):*

   This scheduling algorithm optimizes both dynamic and static energy consumption. The dynamic energy is optimized using the frequency scaling technique used in MCS while the static energy is optimized using shutdown and procrastination techniques.

In all the above algorithms, in case of multi-core processors, the periodic tasks are partitioned and allocated to different cores. The periodic tasks assigned to each core are scheduled using EDF scheduling policy. The aperiodic tasks are assigned to any of the cores and are scheduled along with periodic tasks on that core using TBS. In case of any of the frequency scaling algorithms, the aperiodic tasks are executed at maximum frequency. All the algorithms are capable of scheduling the tasks on uniprocessor as well as multi-core processor platform.

5.3.2.5. Output Subsystem of STREAM

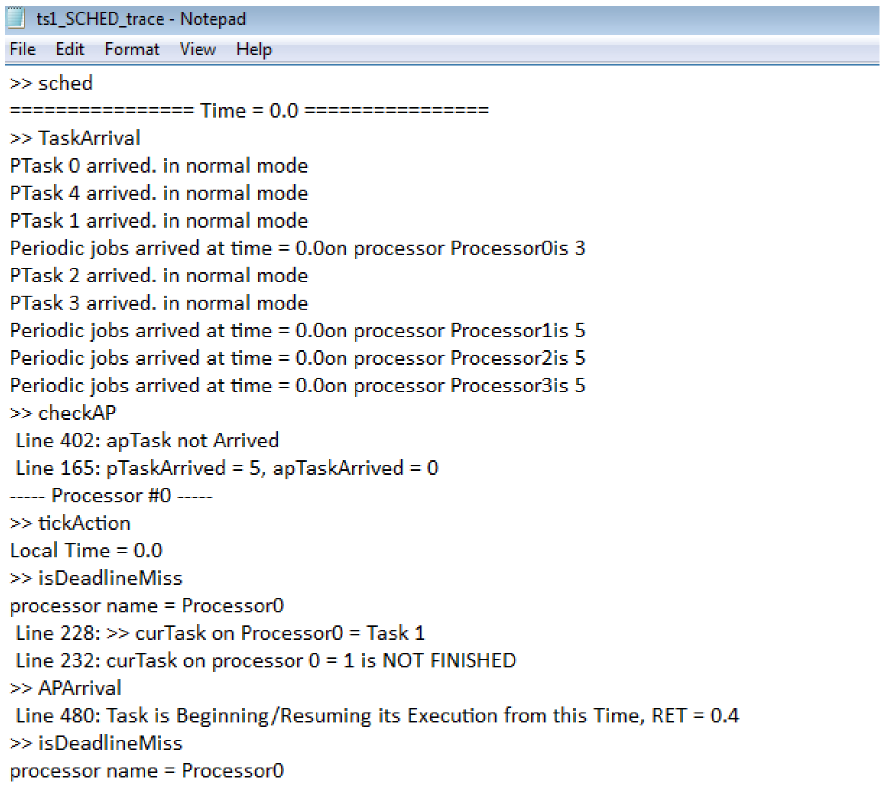
Output subsystem of the simulator provides ways to capture and record every single event over the time of schedule. This subsystem works in background and performs logging of various activities and events that scheduler performs while running. As a result of this, a number of log traces are generated clearly describing the execution history or event history of the scheduler. These log traces are useful in many ways, for instance, to examine and evaluate the correctness and validity of designed scheduler, to verify the execution against the expected outcome, to easily track down all the activities related to particular a event etc. Output subsystem provides following facilities as a part of simulation result set: Scheduler execution log trace, per-core execution log trace and energy measurement log trace.

- *Scheduler Execution Log Trace:*

This log trace gives the complete details about the execution of scheduler. The details are captured for every single time unit until the hyper period. This includes, the information about the series of actions that took place in one time unit of execution; the information about the processing of every single event and its outcome; the information related to reasons illustrating task state transitions; the statistical information related to various data structures such as static queues, per-core ready queues etc. It describes the important information about task set partition across multiple cores. Figure 5.8 shows the sample log trace generated by the simulator while executing energy aware EDF based scheduler.

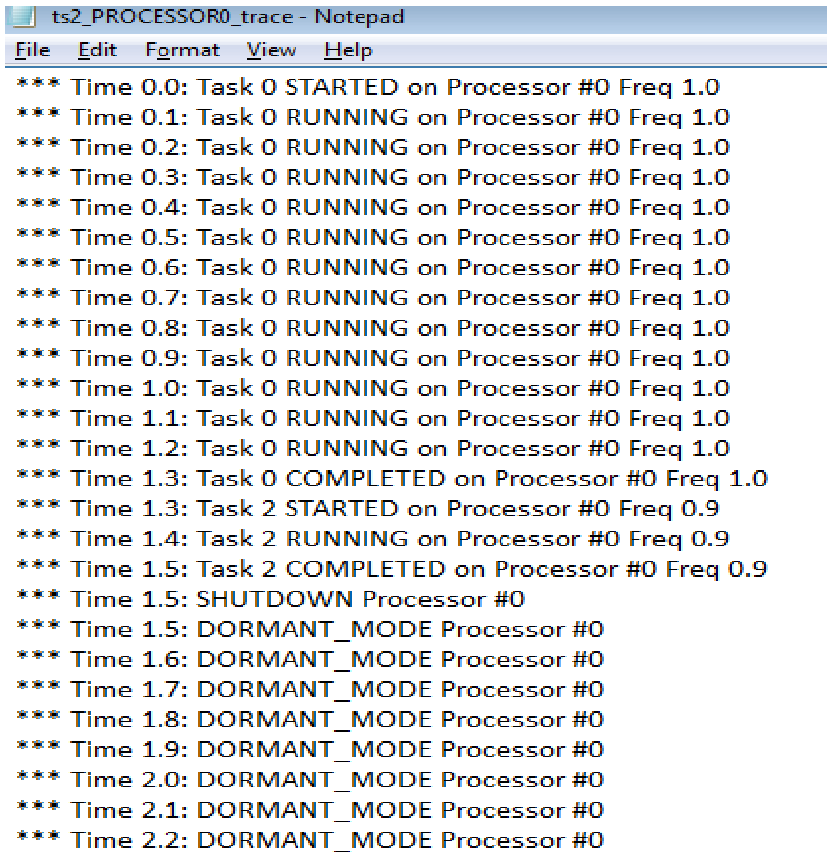
- *Per-core Execution Log Trace:*

This log trace is specifically generated for each core, describing the task execution over one hyper period. In multi-core environment, it is very important to track down the behavioral and statistical information about every single core which is useful in many cases such as, when task migration is allowed across the cores, it is required to know the reason for migration and extract the temporal description about the event. The log trace clearly illustrates, when a particular task is started on the core, when it is finished, preempted or migrated indicating the time of action. It also shows the time action if any task is missing deadline. This is helpful in determining the reason for deadline miss based on the prior execution history, and thereby deciding the cause for deadline miss. Figure 5.9 shows the sample log trace of 2-cores execution in parallel.



```
>> sched
=============== Time = 0.0 ===============
>> TaskArrival
PTask 0 arrived. in normal mode
PTask 4 arrived. in normal mode
PTask 1 arrived. in normal mode
Periodic jobs arrived at time = 0.0on processor Processor0is 3
PTask 2 arrived. in normal mode
PTask 3 arrived. in normal mode
Periodic jobs arrived at time = 0.0on processor Processor1is 5
Periodic jobs arrived at time = 0.0on processor Processor2is 5
Periodic jobs arrived at time = 0.0on processor Processor3is 5
>> checkAP
 Line 402: apTask not Arrived
 Line 165: pTaskArrived = 5, apTaskArrived = 0
----- Processor #0 -----
>> tickAction
Local Time = 0.0
>> isDeadlineMiss
processor name = Processor0
 Line 228: >> curTask on Processor0 = Task 1
 Line 232: curTask on processor 0 = 1 is NOT FINISHED
>> APArrival
 Line 480: Task is Beginning/Resuming its Execution from this Time, RET = 0.4
>> isDeadlineMiss
processor name = Processor0
```

Figure 5.8: A partial snapshot showing scheduler execution log trace

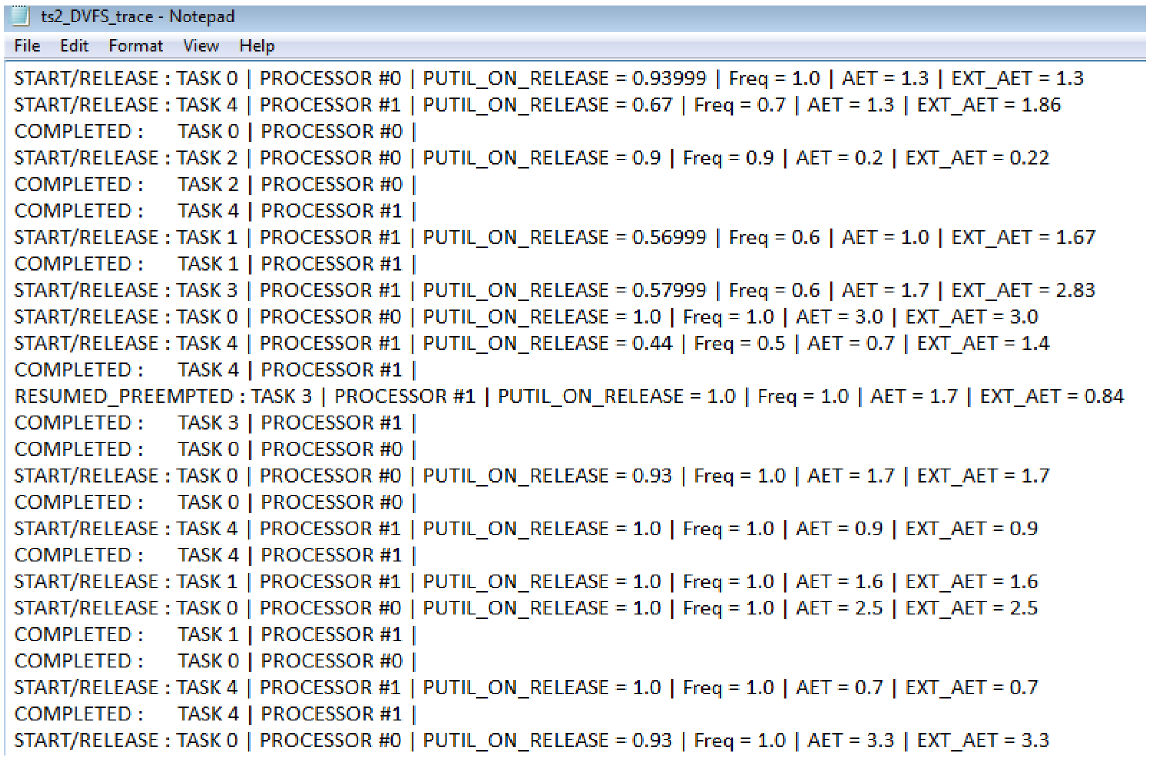


```
*** Time 0.0: Task 0 STARTED on Processor #0 Freq 1.0
*** Time 0.1: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 0.2: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 0.3: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 0.4: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 0.5: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 0.6: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 0.7: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 0.8: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 0.9: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 1.0: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 1.1: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 1.2: Task 0 RUNNING on Processor #0 Freq 1.0
*** Time 1.3: Task 0 COMPLETED on Processor #0 Freq 1.0
*** Time 1.3: Task 2 STARTED on Processor #0 Freq 0.9
*** Time 1.4: Task 2 RUNNING on Processor #0 Freq 0.9
*** Time 1.5: Task 2 COMPLETED on Processor #0 Freq 0.9
*** Time 1.5: SHUTDOWN Processor #0
*** Time 1.5: DORMANT_MODE Processor #0
*** Time 1.6: DORMANT_MODE Processor #0
*** Time 1.7: DORMANT_MODE Processor #0
*** Time 1.8: DORMANT_MODE Processor #0
*** Time 1.9: DORMANT_MODE Processor #0
*** Time 2.0: DORMANT_MODE Processor #0
*** Time 2.1: DORMANT_MODE Processor #0
*** Time 2.2: DORMANT_MODE Processor #0
```

Figure 5.9: A partial snapshot showing parallel per-core execution trace

- *Energy measurement log trace:*

This log trace provides useful information about energy management settings for energy aware schedulers. This is modeled according to the type of energy management technique employed in the scheduler. In case of DVFS technique, the log trace includes the sequential information about frequency-voltage settings done at various events throughout the schedule. This clearly describes the energy variations that occurred in the schedule, which is useful to determine various energy related equations such as energy savings behavior, processor frequency behavior against load etc. Figure 5.10 shows the sample view of DVFS setting log trace generated by simulator out of energy aware scheduler.



```
START/RELEASE : TASK 0 | PROCESSOR #0 | PUTIL_ON_RELEASE = 0.93999 | Freq = 1.0 | AET = 1.3 | EXT_AET = 1.3
START/RELEASE : TASK 4 | PROCESSOR #1 | PUTIL_ON_RELEASE = 0.67 | Freq = 0.7 | AET = 1.3 | EXT_AET = 1.86
COMPLETED :     TASK 0 | PROCESSOR #0 |
START/RELEASE : TASK 2 | PROCESSOR #0 | PUTIL_ON_RELEASE = 0.9 | Freq = 0.9 | AET = 0.2 | EXT_AET = 0.22
COMPLETED :     TASK 2 | PROCESSOR #0 |
COMPLETED :     TASK 4 | PROCESSOR #1 |
START/RELEASE : TASK 1 | PROCESSOR #1 | PUTIL_ON_RELEASE = 0.56999 | Freq = 0.6 | AET = 1.0 | EXT_AET = 1.67
COMPLETED :     TASK 1 | PROCESSOR #1 |
START/RELEASE : TASK 3 | PROCESSOR #1 | PUTIL_ON_RELEASE = 0.57999 | Freq = 0.6 | AET = 1.7 | EXT_AET = 2.83
START/RELEASE : TASK 0 | PROCESSOR #0 | PUTIL_ON_RELEASE = 1.0 | Freq = 1.0 | AET = 3.0 | EXT_AET = 3.0
START/RELEASE : TASK 4 | PROCESSOR #1 | PUTIL_ON_RELEASE = 0.44 | Freq = 0.5 | AET = 0.7 | EXT_AET = 1.4
COMPLETED :     TASK 4 | PROCESSOR #1 |
RESUMED_PREEMPTED : TASK 3 | PROCESSOR #1 | PUTIL_ON_RELEASE = 1.0 | Freq = 1.0 | AET = 1.7 | EXT_AET = 0.84
COMPLETED :     TASK 3 | PROCESSOR #1 |
COMPLETED :     TASK 0 | PROCESSOR #0 |
START/RELEASE : TASK 0 | PROCESSOR #0 | PUTIL_ON_RELEASE = 0.93 | Freq = 1.0 | AET = 1.7 | EXT_AET = 1.7
COMPLETED :     TASK 0 | PROCESSOR #0 |
START/RELEASE : TASK 4 | PROCESSOR #1 | PUTIL_ON_RELEASE = 1.0 | Freq = 1.0 | AET = 0.9 | EXT_AET = 0.9
COMPLETED :     TASK 4 | PROCESSOR #1 |
START/RELEASE : TASK 1 | PROCESSOR #1 | PUTIL_ON_RELEASE = 1.0 | Freq = 1.0 | AET = 1.6 | EXT_AET = 1.6
START/RELEASE : TASK 0 | PROCESSOR #0 | PUTIL_ON_RELEASE = 1.0 | Freq = 1.0 | AET = 2.5 | EXT_AET = 2.5
COMPLETED :     TASK 1 | PROCESSOR #1 |
COMPLETED :     TASK 0 | PROCESSOR #0 |
START/RELEASE : TASK 4 | PROCESSOR #1 | PUTIL_ON_RELEASE = 1.0 | Freq = 1.0 | AET = 0.7 | EXT_AET = 0.7
COMPLETED :     TASK 4 | PROCESSOR #1 |
START/RELEASE : TASK 0 | PROCESSOR #0 | PUTIL_ON_RELEASE = 0.93 | Freq = 1.0 | AET = 3.3 | EXT_AET = 3.3
```

Figure 5.10: A partial snapshot of DVFS setting log trace

5.3.2.6. Performance Analyzer and Scheduling Profiler

This is a very important module particularly when performance study and examination of any scheduler is of top priority. It provides several options to do performance analysis on top of scheduler. It generates analysis information in text format in separate files which can be examined manually or can be supplied to a visual analyzer in order to get more user friendly visual results. Visual analyzer is a part of performance analyzer which works as independent module and uses a rich graphics library to visualize the performance analysis results in a user friendly visual format such as line graphs, bar charts etc. Visual analysis makes it very easy to quickly compare the performance of various scheduling algorithms against each other.

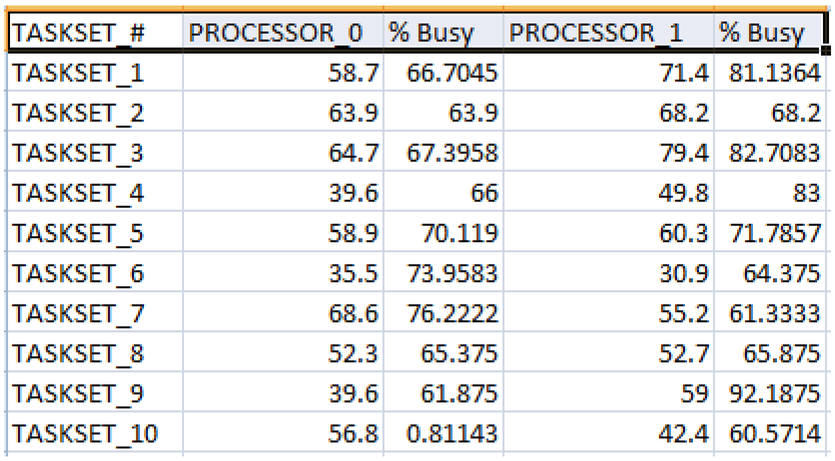
Currently, STREAM provides following performance analysis options.

- Processor Utilization Analysis
- Task Response Time Analysis

- Decision Point Analysis
- Energy Consumption Statistical Analysis

- *Processor Utilization Analysis:*

  Analyzing the utilization statistics of processor is important since it is the major parameter for frequency selection. It is particularly useful in determining the busy/idle percentage of processor against a given task load and partition. Figure 5.11 shows the processor utilization by individual task sets.

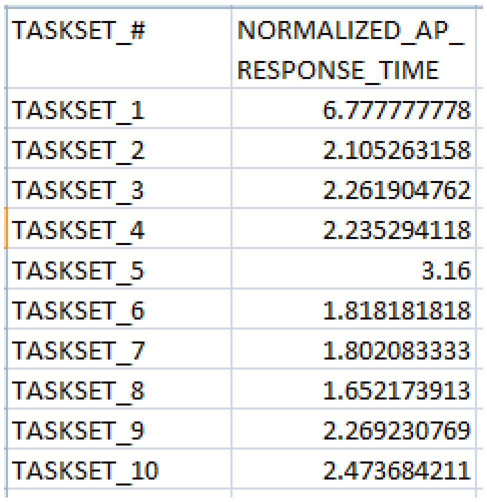


TASKSET_#	PROCESSOR_0	% Busy	PROCESSOR_1	% Busy
TASKSET_1	58.7	66.7045	71.4	81.1364
TASKSET_2	63.9	63.9	68.2	68.2
TASKSET_3	64.7	67.3958	79.4	82.7083
TASKSET_4	39.6	66	49.8	83
TASKSET_5	58.9	70.119	60.3	71.7857
TASKSET_6	35.5	73.9583	30.9	64.375
TASKSET_7	68.6	76.2222	55.2	61.3333
TASKSET_8	52.3	65.375	52.7	65.875
TASKSET_9	39.6	61.875	59	92.1875
TASKSET_10	56.8	0.81143	42.4	60.5714

Figure 5.11: Snapshot showing per core utilization statistics

- *Task Response Time Analysis:*

Some schedulers are designed with a view to minimize the response time of a task. Minimum response time guarantees are particularly stringent for aperiodic tasks since they arrive at random time and are generally responsible for performing a high priority action which needs quick attention. This objective is achieved while ensuring the deadline guarantees of periodic tasks. STREAM provides the facility to determine the responsiveness of the task by calculating its response time over a span of execution in one hyper period. In many schedulers, in order to minimize the response time, optimization decisions such as task migration etc. are taken. Tasks are generally migrated to a least utilized core, thereby allowing that task to get an early chance of execution and hence finish early. In STREAM, many schedulers employ this technique for aperiodic tasks execution. The procedure to calculate the response time of aperiodic tasks is to first record the arrival and finish time for every aperiodic task. At the end of one hyper period, analyzer takes charge and calculates the response time of all aperiodic tasks. In order to make the analysis easy to examine, response time of a task is normalized over its actual execution time. Finally, the maximum response time value among all aperiodic tasks in a task set is selected for stating the analytical result. Figure 5.12 shows the partial snapshot of an output file which stores the output data corresponding to normalized response time values generated by STREAM per task set.

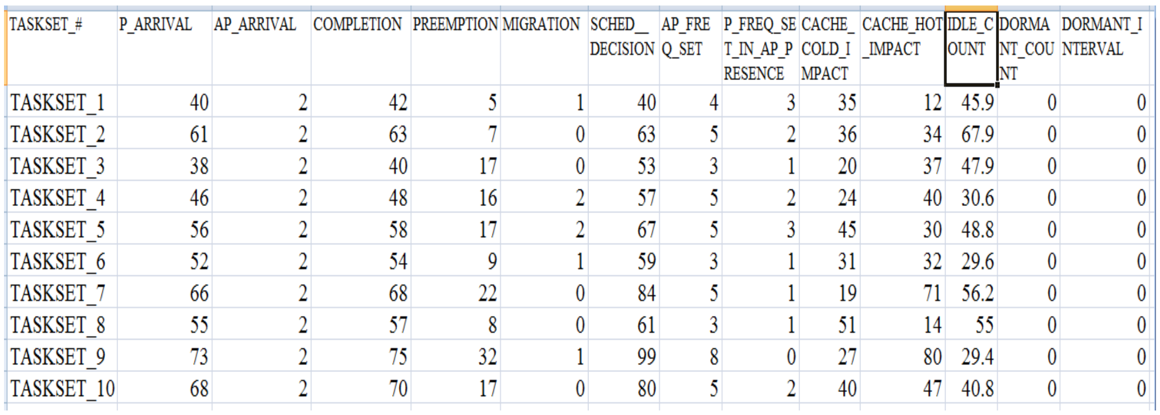


TASKSET_#	NORMALIZED_AP_ RESPONSE_TIME
TASKSET_1	6.777777778
TASKSET_2	2.105263158
TASKSET_3	2.261904762
TASKSET_4	2.235294118
TASKSET_5	3.16
TASKSET_6	1.818181818
TASKSET_7	1.802083333
TASKSET_8	1.652173913
TASKSET_9	2.269230769
TASKSET_10	2.473684211

Figure 5.12: Snapshot showing normalized response time values generated by STREAM per task set

- *Decision Point Analysis:*

This analysis is performed in order to examine the behavior of task execution states, to know the statistical information about various task state transitions, and to record the accounting information about them. As a part of this analysis, analyzer maintains and updates various decision counts corresponding to various task scheduling events as shown in figure 5.13. This includes arrival points, preemption points, completion points, migration points, hot and cold cache impact points, scheduling points, idle count, dormant count and dormant intervals over one hyper period. These counts are very useful is analyzing the performance of a scheduling algorithm with respect to other algorithms.



TASKSET_#	P_ARRIVAL	AP_ARRIVAL	COMPLETION	PREEMPTION	MIGRATION	SCHED__DECISION	AP_FREQ_SET	P_FREQ_SET_IN_AP_PRESENCE	CACHE_COLD_IMPACT	CACHE_HOT_IMPACT	IDLE_COUNT	DORMANT_COUNT	DORMANT_INTERVAL
TASKSET_1	40	2	42	5	1	40	4	3	35	12	45.9	0	0
TASKSET_2	61	2	63	7	0	63	5	2	36	34	67.9	0	0
TASKSET_3	38	2	40	17	0	53	3	1	20	37	47.9	0	0
TASKSET_4	46	2	48	16	2	57	5	2	24	40	30.6	0	0
TASKSET_5	56	2	58	17	2	67	5	3	45	30	48.8	0	0
TASKSET_6	52	2	54	9	1	59	3	1	31	32	29.6	0	0
TASKSET_7	66	2	68	22	0	84	5	1	19	71	56.2	0	0
TASKSET_8	55	2	57	8	0	61	3	1	51	14	55	0	0
TASKSET_9	73	2	75	32	1	99	8	0	27	80	29.4	0	0
TASKSET_10	68	2	70	17	0	80	5	2	40	47	40.8	0	0

Figure 5.13: Snapshot showing different decision counts per task set

The impact of cache memory access on energy consumption is analyzed by measuring the number of hot and cold cache impact points. A separate cache impact detector module is designed in STREAM in order to detect cache impact points. This module is embedded inside the processor and executes along with processor. There are two types of cache impacts considered in this analysis.

- *Hot Cache Impact:*

Hot cache impact occurs when the same task resumes its execution after a delay of *k*-higher priority task(s) execution after preemption. Here the value of *k* depends on the type of processor model followed. In case of synthesized processor model, generally a value of *k* is taken as 1. This means that a task remains available in cache if it resumes its execution after a delay of one higher priority task execution after preemption. This is said to have a hot impact since processor can avoid traversing the memory hierarchy to get that task thereby reducing the access overhead.

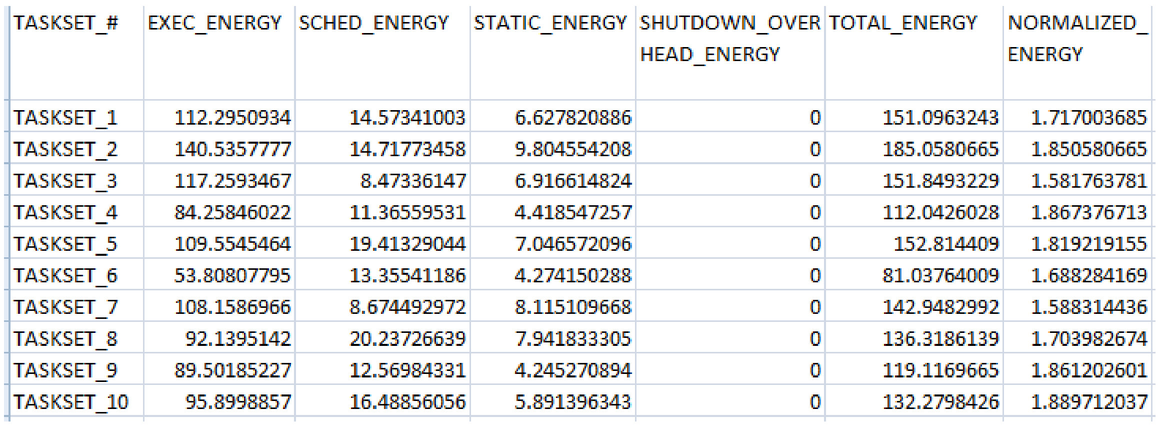
- *Cold Cache Impact:*

If a task doesn't resume its execution after a delay of *k*-higher priority tasks execution after preemption, then processor has to traverse the memory hierarchy in order to bring that task back to the cache, which obviously increases the access overhead. This effect is termed as cold cache impact.

In STREAM, separate counts are maintained for detecting hot impact and cold impact events. These counts are generated as a part of decision point counts as shown in last columns of figure 5.13.

- *Energy Consumption Statistical Analysis:*

This type of analysis is very useful when a performance of energy aware scheduler is to be measured in terms of its total energy consumption. STREAM provides the facility to calculate the energy consumptions as guided by a specified energy controller by following the energy profile embedded inside the processor. The two sources of energy consumption: dynamic energy consumption and static energy consumption are measured and used for the analysis of various energy aware scheduling algorithms. Figure 5.14 shows the snap shot of the analytical results recorded as part of the energy consumption statistical analysis.



TASKSET_#	EXEC_ENERGY	SCHED_ENERGY	STATIC_ENERGY	SHUTDOWN_OVERHEAD_ENERGY	TOTAL_ENERGY	NORMALIZED_ENERGY
TASKSET_1	112.2950934	14.57341003	6.627820886	0	151.0963243	1.717003685
TASKSET_2	140.5357777	14.71773458	9.804554208	0	185.0580665	1.850580665
TASKSET_3	117.2593467	8.47336147	6.916614824	0	151.8493229	1.581763781
TASKSET_4	84.25846022	11.36559531	4.418547257	0	112.0426028	1.867376713
TASKSET_5	109.5545464	19.41329044	7.046572096	0	152.814409	1.819219155
TASKSET_6	53.80807795	13.35541186	4.274150288	0	81.03764009	1.688284169
TASKSET_7	108.1586966	8.674492972	8.115109668	0	142.9482992	1.588314436
TASKSET_8	92.1395142	20.23726639	7.941833305	0	136.3186139	1.703982674
TASKSET_9	89.50185227	12.56984331	4.245270894	0	119.1169665	1.861202601
TASKSET_10	95.8998857	16.48856056	5.891396343	0	132.2798426	1.889712037

Figure 5.14: Snapshot showing energy consumption per task set

5.3.2.7. Visual Analyzer

A visual analyzer is designed as a part of performance analyzer. It is responsible for the visual display of the analysis results obtained by performance analyzer. It uses a rich graphics and chart library called XCHART to convert the analysis results into various types of graphs such as series graph, bar charts etc. There is also a provision inside visual analyzer to compare the analysis results of different schedulers by plotting the comparison graphs. This helps to quickly learn and examine the behavior of different schedulers under various aspects. In the current version of STREAM, visual analyzer can plot different types of graphs such as energy consumption analysis plot, response time analysis plot for various types of inputs: for different number of periodic tasks in a task set, for increasing periodic utilization etc. Figure 5.15 shows a line graph generated using visual analyzer of STREAM that shows the energy consumption measured by using four different scheduling algorithms against increasing periodic utilization.

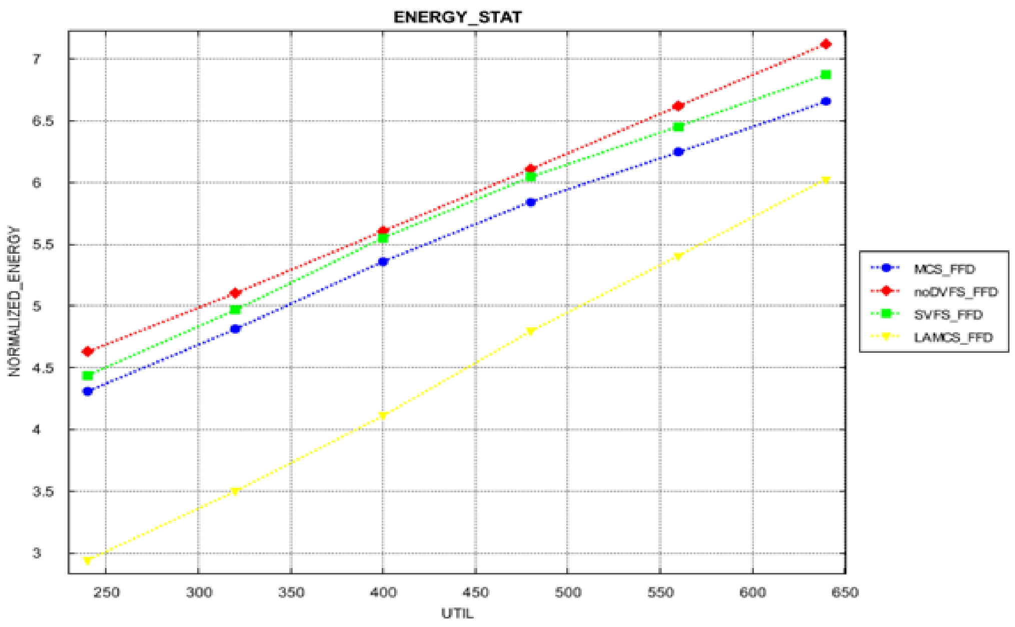


Figure 5.15: A line graph showing normalized energy consumption Vs. periodic utilization

5.4. Summary

Simulation is an important and one of the well accepted methods for the validation of scheduling algorithms in real time systems community. This chapter discusses the design and implementation aspects of one such scheduler simulator called STREAM. This simulator is designed and developed as the part of the work in this thesis for the evaluation and validation of the proposed schedulers.

The main focus of this simulator is to develop the energy efficient real time schedulers for the mixed task model. It is written in java programming language that makes use of object oriented paradigm. The current implementation of STREAM includes various EDF based energy aware and non energy aware schedulers. The output generated by the simulator includes various log traces that can help the programmer to test and validate the working of the algorithms. In order to examine the performance of the new scheduler against any existing one, various performance parameters such as energy consumption per core, processor utilization, decision points etc. can be generated in different file formats. Visual analyzer tool is an attractive feature of STREAM which can generate graphs given the required performance related data as input.

Chapter 6

Conclusions and Future Work

This chapter summarizes the main contributions and significant results obtained in this research work along with insight into the future extensions of the work. This thesis addresses the issue of overall energy optimization in multi-core systems at the operating system level using efficient real time task scheduling algorithms. The objectives are achieved by proposing various energy efficient real time task scheduling algorithms to minimize dynamic as well as static energy consumption of the uniprocessor as well as multi-core processors. A full-fledged scheduler simulation tool is developed to implement and test various non-energy aware and energy aware real time task scheduling algorithms.

The dynamic energy optimization of the uniprocessor and multi-core processor is achieved by dynamic voltage and frequency scaling techniques whereas the static energy optimization is achieved by dynamic shutdown and procrastination techniques. The timing constraints of hard real time tasks and responsiveness of soft real time tasks are taken care while optimizing overall energy consumption. The response time of the soft real time tasks is a significant parameter which is handled by the efficient task allocation strategies and use of efficient aperiodic scheduling policy, TBS. In Chapter 2, we summarized the major work done in the area of energy efficient scheduling algorithms for uniprocessor, multiprocessor and multi-core processors for hard and soft real time tasks. It was observed in literature survey that the issue of energy efficiency at operating system level is sufficiently addressed for uniprocessor platform. For multiprocessor and multi-core processor platform, even though there exist many energy efficient scheduling techniques that optimize dynamic and static energy consumption, majority of them can schedule only hard real time tasks and cannot handle hard and soft real time tasks together. Therefore, there is a need to develop a complete scheduling framework which can schedule both hard and soft real time tasks together, optimize both dynamic and static energy consumption and ensure the responsiveness of soft real time tasks without hampering the timeliness of hard real time tasks. In Chapter 3 we proposed algorithms that optimized dynamic energy of uniprocessor and multi-core processors. In Chapter 4 we focused on optimization of dynamic as well as static energy of multi-core processors. Finally, in Chapter 5 we described in detail the design and development of the simulation tool that is capable of running and testing all the proposed algorithms.

In chapter 3 of this thesis, we focused on dynamic energy optimization by using real time scheduling algorithms for mixed task sets that contain periodic as well as aperiodic tasks on uniprocessor and homogeneous multi-core processor. We have

- ⇨ Proposedd two energy efficient real time scheduling algorithms for uniprocessor platform, EEDVFS and EE-UCS that use EDF scheduling for periodic tasks and two different bandwidth preserving algorithms, DS and TBS for scheduling aperiodic tasks respectively. EE-UCS performs better than EEDVFS in terms of responsiveness of aperiodic tasks and therefore, in multi-core scheduling, TBS is used for aperiodic task scheduling.
- ⇨ Proposed an energy efficient real time task scheduling algorithm, MCS, which optimizes dynamic energy of each processing core in multi-core environment and optimizes the response time of aperiodic tasks. MCS works in two steps: task allocation and task scheduling. Static task allocation method is followed for periodic tasks while arbitrarily arriving aperiodic tasks are dynamically allocated to the least loaded processor core. This hybrid task allocation approach efficiently utilizes the remaining processing load of each processing core by scheduling aperiodic tasks globally to the lightly loaded processor core resulting in increase in overall processor utilization. The periodic and aperiodic tasks assigned to each core are independently scheduled on respective cores.
- ⇨ Used the real time scheduling algorithms, namely, EDF and TBS to schedule the mixed task sets. The proposed method of core's dynamic utilization calculation allowed the use of available slack time for dynamic frequency and voltage scaling resulting in significant reduction in dynamic energy of individual core.
- ⇨ Implemented and tested EEDVFS, EE-UCS and MCS algorithms. The analysis of these algorithms is done on various parameters like energy consumption, aperiodic response time, preemption count, scheduling decision points, migration count etc.

The behavior of the algorithms is tested on these parameters by varying number of periodic tasks, number of aperiodic tasks, number of processing cores and by increasing total utilization of the periodic tasks in a task set. For the testing of each performance metric, 100 runs were made on 100 different synthetic task sets.

⇨ Analyzed the simulation results and it is observed that, MCS performs better than all the other algorithms used for comparison. It significantly reduces energy consumption compared to non-energy aware scheduling algorithm. It also gives better results than the existing energy efficient cycle conserving scheduling algorithm. There is little overhead in terms of preemptions, migrations and other scheduling events that consume some energy. The aperiodic response time achieved by MCS is nearly equal to the optimal value of 1. The proposed algorithm MCS saves 29.4%, 10.1% and 8.9% energy as compared to Non-DVFS, SVFS and cycle conserving algorithms respectively over one hyper period. This states that over multiple hyper periods, MCS shows significant amount of energy saving.

The limitation of DVFS based technique is that if we reduce the frequency below the critical speed, the static energy consumption increases, and as a result, it increases overall energy consumption. In the above work, although, the frequency is not reduced below critical speed but static energy optimization is not explicitly taken care. We have proposed a new algorithm that optimizes both dynamic and static energy consumption which is discussed in chapter 4.

In chapter 4 of this thesis, we have focused on the overall energy optimization that includes the optimization of dynamic and static energy. The energy optimization is performed by using real time scheduling algorithm for mixed task sets that contain periodic as well as aperiodic tasks on homogeneous multi-core processor. We have

⇨ Proposed and implemented a leakage aware DVFS based real time scheduling algorithm, LAMCS, to optimize dynamic as well as static energy consumption of each processing core in a multi-core environment. It also optimizes the response time of aperiodic tasks by meeting the hard deadlines of the periodic tasks. LAMCS is sub-divided into two parts: (1) Task Allocation (2) Task Scheduling. Task allocation is done in similar way as that used in MCS.

- ⇨ Done scheduling of tasks (using EDF and TBS algorithms) which takes care of optimizing dynamic energy consumption using DVFS based technique and static energy consumption using dynamic shutdown and procrastination techniques. It does not allow frequency scaling below critical speed. In order to reduce the shutdown overhead, it tries to extend the shutdown intervals using procrastination technique. Procrastination technique delays the execution of a task until the processor wakes up.
- ⇨ Handled the aperiodic tasks in the same way as in MCS except that if the task arrives during the shutdown interval, its virtual deadline is calculated with respect to wake up time.
- ⇨ Implemented LAMCS in STREAM simulator and analysis is done on various parameters such as energy consumption, aperiodic response time, preemption count, scheduling decision points, migration count etc. The behavior of the algorithm is tested on these parameters by varying number of periodic tasks, number of aperiodic tasks, number of processing cores and by increasing total utilization of the periodic tasks in a task set. For each performance metric, 100 runs were made on 100 different randomly generated task sets.
- ⇨ On the basis of the simulation results, it is observed that, in case of FFD partitioning, LAMCS gives significant energy saving as compared to non energy aware scheduling algorithms, SVFS and MCS. It achieves 25.48%, 32.1% and 27.89% energy saving as compared to MCS, Non-DVFS and SVFS algorithms respectively. There is little overhead in terms of preemptions, migrations and other scheduling events that consume some energy. The aperiodic response time achieved by LAMCS in case of FFD partitioning scheme is more as compared to others. In case of WFD partitioning scheme, LAMCS consumes nearly equal amount of energy when compared with MCS resulting in less energy saving. In case of WFD partitioning scheme, LAMCS achieves 1.51%, 29.69% and 10.15% energy saving as compared to MCS, Non-DVFS and SVFS algorithms respectively. There is small percentage increase in aperiodic response time as compared to other algorithms. However, all the algorithms including LAMCS achieve nearly optimal response time with WFD partitioning scheme. The limitation of LAMCS is that in case of

WFD partitioning scheme, the number of shutdown intervals are more and as a result, energy consumed in shutting down and waking up the processor core is significant.

Simulation is an important and one of the well accepted methods for the validation of scheduling algorithms in real time systems community. Chapter 5 discusses the design and implementation aspects of scheduler/simulator developed by us called STREAM. In this thesis we have

- ⇨ Designed and developed a simulator called STREAM for the evaluation and validation of the proposed schedulers. The main focus of this simulator is to develop the energy efficient real time schedulers for the mixed task model. It is written in java programming language that makes use of object oriented paradigm. The current implementation of STREAM includes various EDF based energy aware and non energy aware schedulers. The output generated by the simulator includes various log traces that can help the programmer to test and validate the working of the algorithms. In order to examine the performance of the new scheduler against the existing one, various performance parameters such as energy consumption per core, processor utilization, decision points etc. can be generated in different file formats. Visual analyzer tool is an attractive feature of STREAM which can generate graphs given the required performance related data as input.

Directions for Future Work

- ⇨ The proposed energy efficient scheduling algorithms can be extended by relaxing some of the assumptions like homogeneous multi-core to heterogeneous multi-core processors, individual clock based processors to voltage-island based multi-core processors.
- ⇨ The proposed energy efficient scheduling algorithms can be further extended to schedule sporadic as well as mixed criticality tasks as wide variety of real time embedded applications make use of such types of tasks.
- ⇨ The proposed energy efficient scheduling algorithms can be applied in robotic control and vehicular networks used in self driving vehicles on roads.

- The proposed scheduling algorithms can be integrated on any real time kernel such as RTLinux.
- In case of static energy optimization, further improvement can be done to reduce the number of idle intervals so that the shutdown overheads can be further reduced.
- System wide energy consumption such as energy consumed by memory sub-systems, interconnection network etc can be addressed.

Appendix A

STREAM: Simulation Tool for Real time Energy efficient scheduling and Analysis for Multi-core processors

User Manual

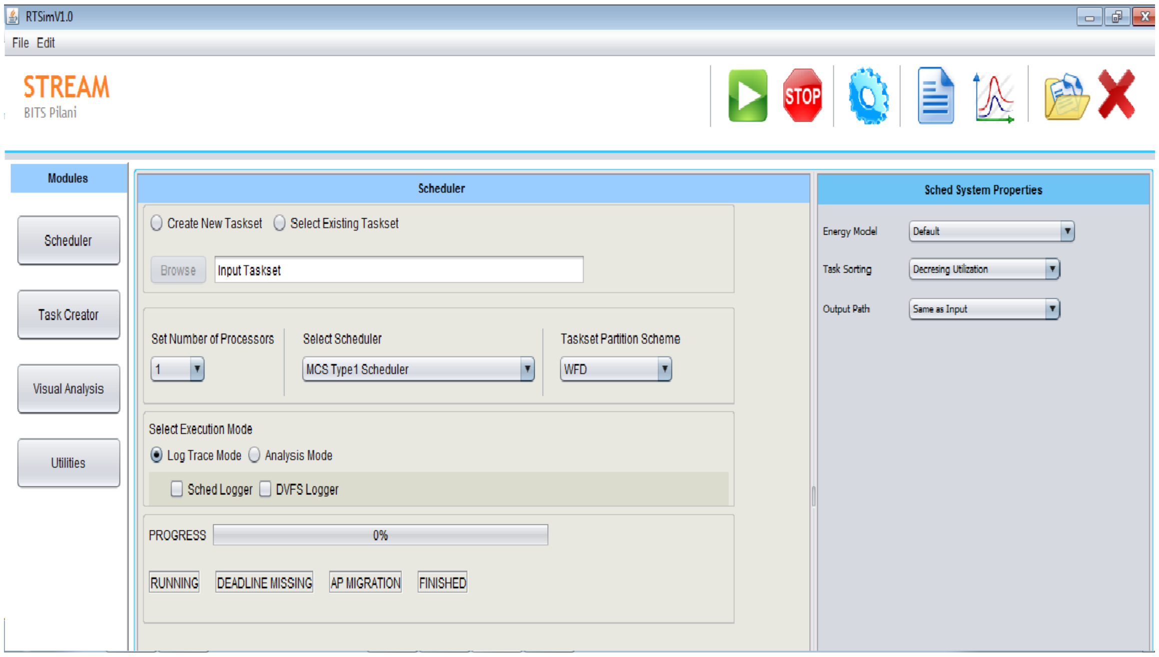
This section explains the step wise details of the working of STREAM simulation tool which includes working of various schedulers, task set generator and performance analyzer.

A.1 Introduction

The simulation tool STREAM is capable of executing various non-energy aware and energy aware real time task scheduling algorithms, generating synthetic real time tasks and generating various outputs such as log traces, various performance metrics such energy consumption, response time, processor busy/idle time, decision counts etc. The simulator is completely based on graphical user interface. The first screen of the simulator is as shown in figure A.1.

A.2 Task Set Generator

The simulation tool STREAM facilitates to generate synthetic task sets. The task sets containing only periodic tasks and mixed task sets containing periodic and aperiodic tasks can be generated using task set generator (or Task Creator). The number of periodic and aperiodic tasks, number of processor cores, total periodic utilization, total aperiodic utilization, range of hyper period and number of task sets to be generated are given as input from the user. In addition to these inputs, there is a provision to select a task generation algorithm and the output path where the generated task sets are stored by the simulator. Each task set contains the tasks with their arrival time, period, worst case execution time and actual execution time. Each task set is also associated with its hyper period. The snapshot of a task set creator in STREAM is shown in figure A.2.



Create New Taskset / Select Existing Taskset

Input Taskset

Set Number of Processors: 1

Select Scheduler: MCS Type1 Scheduler

Taskset Partition Scheme: WFD

Select Execution Mode: [✓ Log Trace Mode] Analysis Mode

Sched Logger / DVFS Logger

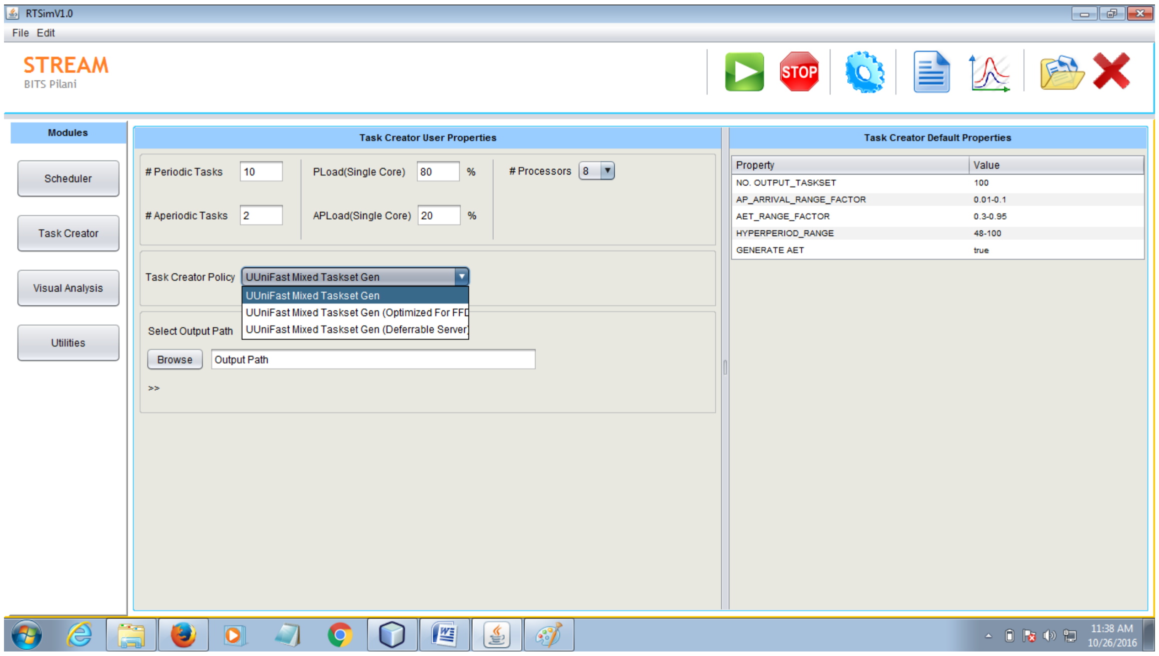
PROGRESS 0%

Energy Model: Default

Task Sorting: Decresing Utilization

Output Path: Same as Input

Figure A.1: Snapshot of Main Screen of STREAM



Periodic Tasks: 10

PLoad(Single Core): 80 %

Processors: 8

Aperiodic Tasks: 2

APLoad(Single Core): 20 %

Task Creator Policy: UUniFast Mixed Taskset Gen

UUniFast Mixed Taskset Gen
UUniFast Mixed Taskset Gen (Optimized For FFD
UUniFast Mixed Taskset Gen (Deferrable Server

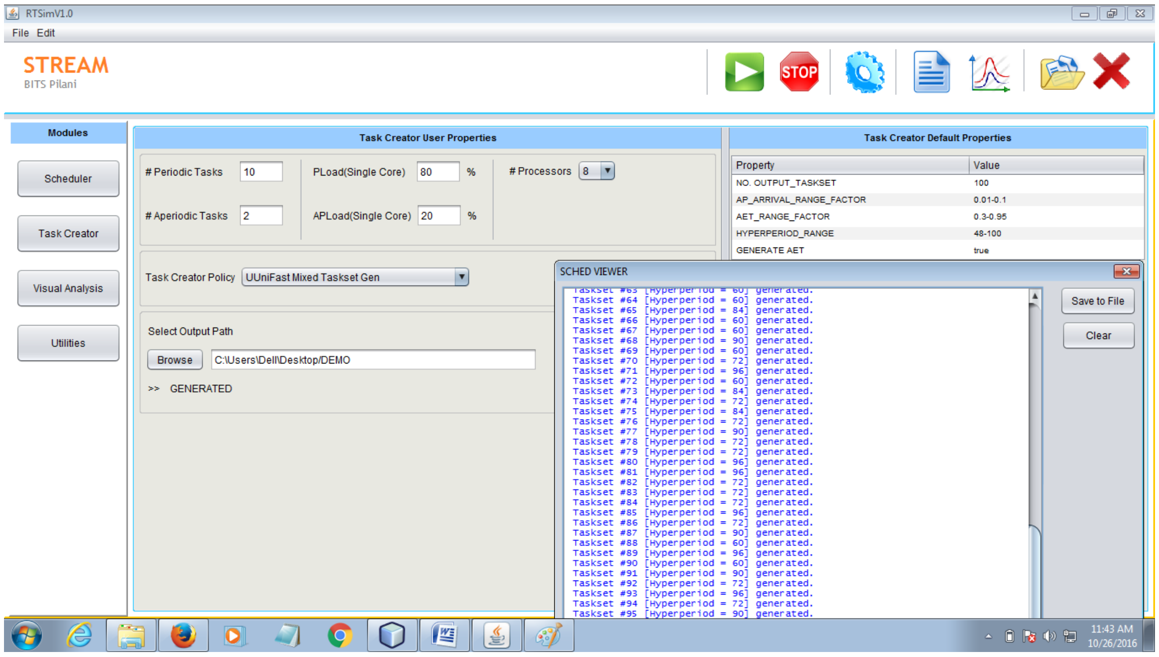
Select Output Path

Output Path

Property	Value
NO. OUTPUT_TASKSET	100
AP_ARRIVAL_RANGE_FACTOR	0.01-0.1
AET_RANGE_FACTOR	0.3-0.95
HYPERPERIOD_RANGE	48-100
GENERATE AET	true

Figure A.2: Snapshot of a Task Generator

There is a provision to view the generation of task sets using SCHED VIEWER as shown in figure A.3.



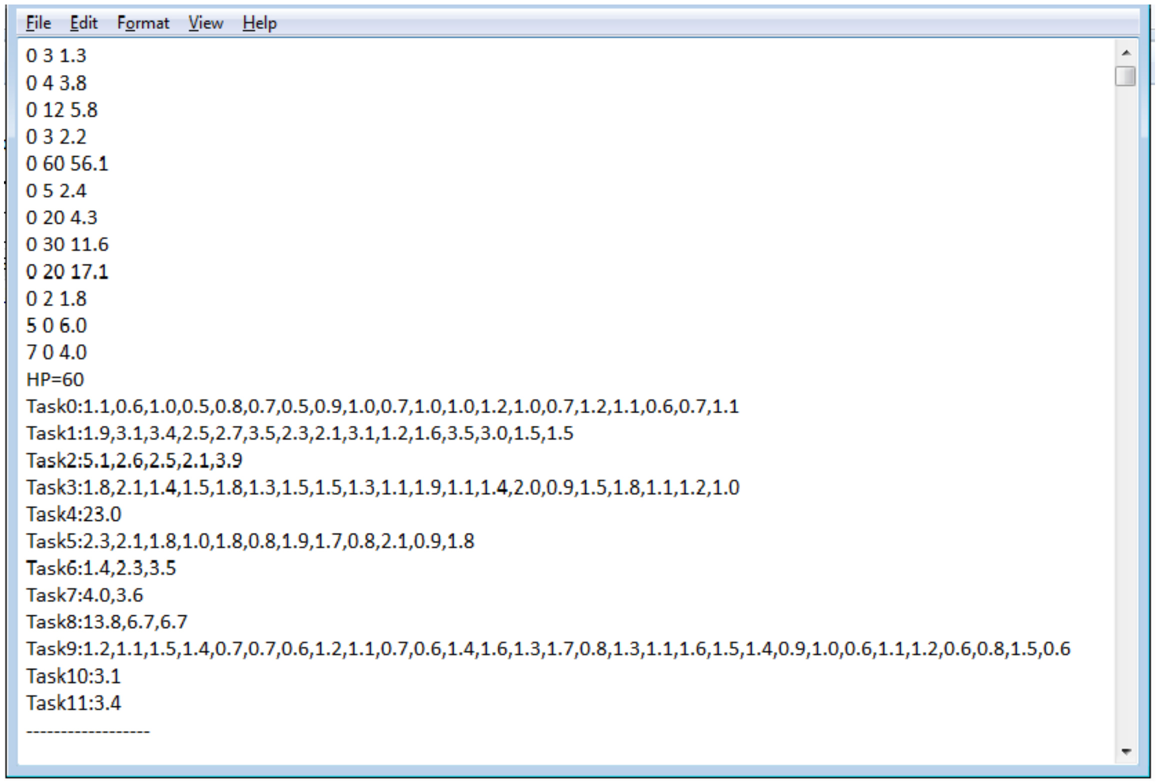
Periodic Tasks: 10
PLoad(Single Core): 80 %
Processors: 8
Aperiodic Tasks: 2
APLoad(Single Core): 20 %
Task Creator Policy: UUniFast Mixed Taskset Gen
Select Output Path: C:\Users\Dell\Desktop/DEMO
>> GENERATED

Property	Value
NO. OUTPUT_TASKSET	100
AP_ARRIVAL_RANGE_FACTOR	0.01-0.1
AET_RANGE_FACTOR	0.3-0.95
HYPERPERIOD_RANGE	48-100
GENERATE AET	true

```
Taskset #63 [Hyperperiod = 60] generated.
Taskset #64 [Hyperperiod = 60] generated.
Taskset #65 [Hyperperiod = 84] generated.
Taskset #66 [Hyperperiod = 60] generated.
Taskset #67 [Hyperperiod = 60] generated.
Taskset #68 [Hyperperiod = 90] generated.
Taskset #69 [Hyperperiod = 60] generated.
Taskset #70 [Hyperperiod = 72] generated.
Taskset #71 [Hyperperiod = 96] generated.
Taskset #72 [Hyperperiod = 60] generated.
Taskset #73 [Hyperperiod = 84] generated.
Taskset #74 [Hyperperiod = 72] generated.
Taskset #75 [Hyperperiod = 84] generated.
Taskset #76 [Hyperperiod = 72] generated.
Taskset #77 [Hyperperiod = 90] generated.
Taskset #78 [Hyperperiod = 72] generated.
Taskset #79 [Hyperperiod = 72] generated.
Taskset #80 [Hyperperiod = 96] generated.
Taskset #81 [Hyperperiod = 96] generated.
Taskset #82 [Hyperperiod = 72] generated.
Taskset #83 [Hyperperiod = 72] generated.
Taskset #84 [Hyperperiod = 72] generated.
Taskset #85 [Hyperperiod = 96] generated.
Taskset #86 [Hyperperiod = 72] generated.
Taskset #87 [Hyperperiod = 90] generated.
Taskset #88 [Hyperperiod = 60] generated.
Taskset #89 [Hyperperiod = 96] generated.
Taskset #90 [Hyperperiod = 60] generated.
Taskset #91 [Hyperperiod = 90] generated.
Taskset #92 [Hyperperiod = 72] generated.
Taskset #93 [Hyperperiod = 96] generated.
Taskset #94 [Hyperperiod = 72] generated.
Taskset #95 [Hyperperiod = 90] generated.
```

Figure A.3. Snapshot showing generation of task sets in SCHED VIEWER

The text file generated by the task set generator contains the predefined number of task sets. Each task set in the file is separated by a dotted line. At the beginning of each task set, the periodic tasks are written followed by aperiodic tasks and hyper period and at the end actual execution time of each task is mentioned. The attributes of the tasks are ordered as arrival time, *wcet* and period in each line. The snapshot of a file produced by task set generator is shown in figure A.4.



```
0 3 1.3
0 4 3.8
0 12 5.8
0 3 2.2
0 60 56.1
0 5 2.4
0 20 4.3
0 30 11.6
0 20 17.1
0 2 1.8
5 0 6.0
7 0 4.0
HP=60
Task0:1.1,0.6,1.0,0.5,0.8,0.7,0.5,0.9,1.0,0.7,1.0,1.0,1.2,1.0,0.7,1.2,1.1,0.6,0.7,1.1
Task1:1.9,3.1,3.4,2.5,2.7,3.5,2.3,2.1,3.1,1.2,1.6,3.5,3.0,1.5,1.5
Task2:5.1,2.6,2.5,2.1,3.9
Task3:1.8,2.1,1.4,1.5,1.8,1.3,1.5,1.5,1.3,1.1,1.9,1.1,1.4,2.0,0.9,1.5,1.8,1.1,1.2,1.0
Task4:23.0
Task5:2.3,2.1,1.8,1.0,1.8,0.8,1.9,1.7,0.8,2.1,0.9,1.8
Task6:1.4,2.3,3.5
Task7:4.0,3.6
Task8:13.8,6.7,6.7
Task9:1.2,1.1,1.5,1.4,0.7,0.7,0.6,1.2,1.1,0.7,0.6,1.4,1.6,1.3,1.7,0.8,1.3,1.1,1.6,1.5,1.4,0.9,1.0,0.6,1.1,1.2,0.6,0.8,1.5,0.6
Task10:3.1
Task11:3.4
------------------
```

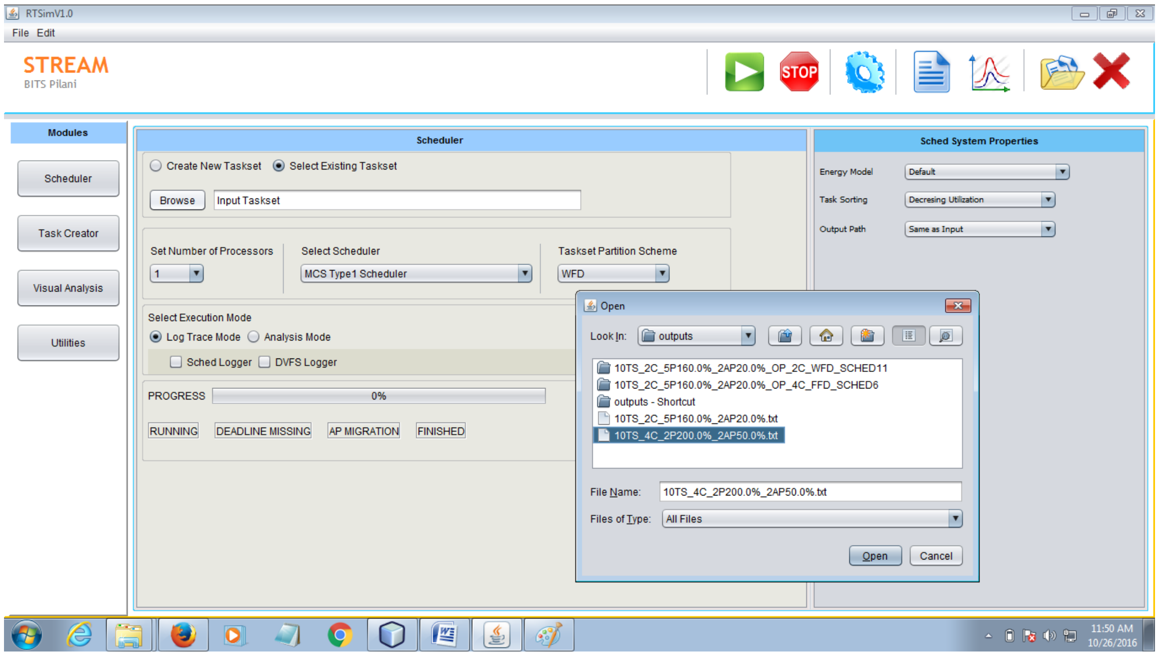
Figure A.4: Sample Task Sets File

A.3 Scheduler

The scheduler interface facilitates the working of various real time schedulers. It takes the input as a text file containing predefined number of task sets and executes each task set one by one. The output of all the task sets is generated and copied in output files. The user has to first create an input file using task creator and then browse the input file in the scheduler interface. Once the input file is selected, user has to select the number of processor cores, the scheduling algorithm, task partitioning algorithm and the type of output user wants to generate.

There are two execution modes - Log trace mode and Analysis mode. In log trace mode two types of log files can be produced. One is *sched logger* for generating the scheduler log which contains the status of different jobs in execution. Another is *DVFS logger* for generating the log which contains the information about the frequency selection at various scheduling points. The other execution mode is Analysis mode in which user can produce different outputs which can be used for analyzing the performance of the scheduler. The various outputs produced in this mode are processor busy/idle time, various decision points like preemption count, migration count, arrival count, cache impact points etc. In addition to these parameters, it can also output the most important performance parameters: energy consumption and aperiodic response time.

There is a progress bar that shows how many tasks have completed the execution and how many are left. It also shows the status of currently executing job by highlighting the boxes below the status bar. For example, if job is running, RUNNING will be highlighted or if the job is migrated to other core, AP MIGRATION is highlighted. Similarly, DEADLINE MISSING and FINISHED will be highlighted. The sequence of steps for executing any scheduling algorithm is shown in figures A.5 to A.13.



Create New Taskset [✓ Select Existing Taskset]

Input Taskset

Set Number of Processors: 1

Select Scheduler: MCS Type1 Scheduler

Taskset Partition Scheme: WFD

Select Execution Mode: [✓ Log Trace Mode] Analysis Mode

Sched Logger DVFS Logger

PROGRESS 0%

RUNNING DEADLINE MISSING AP MIGRATION FINISHED

Energy Model: Default

Task Sorting: Decresing Utilization

Output Path: Same as Input

Open

Look In: outputs

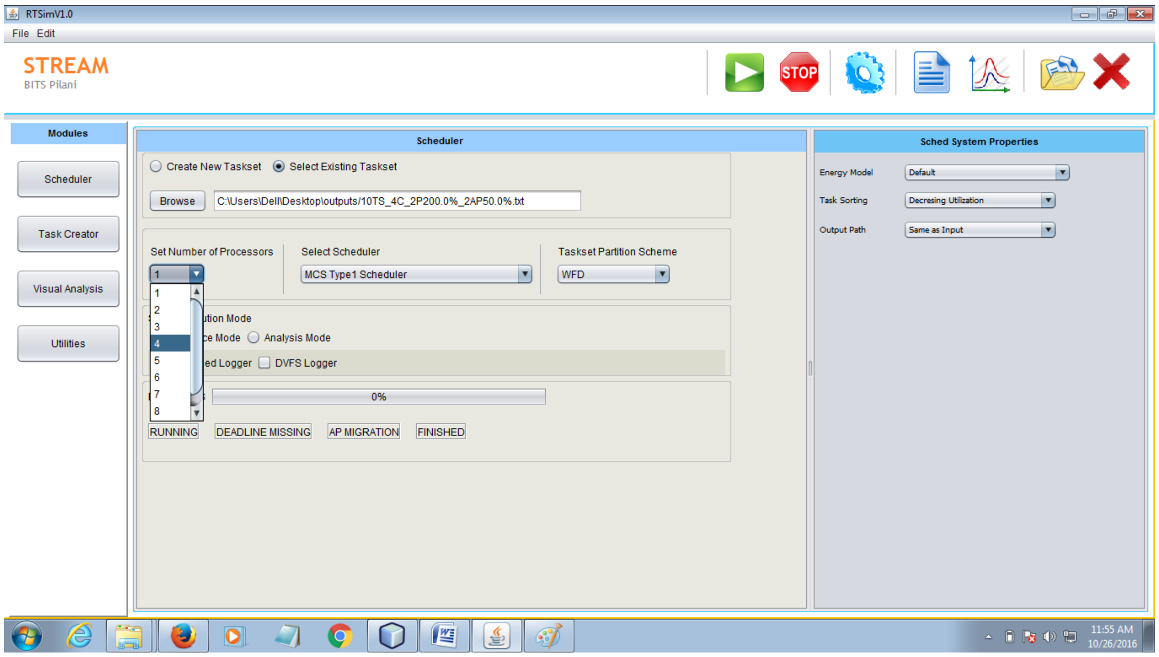
10TS_2C_5P160.0%_2AP20.0%_OP_2C_WFD_SCHED11
10TS_2C_5P160.0%_2AP20.0%_OP_4C_FFD_SCHED6
outputs - Shortcut
10TS_2C_5P160.0%_2AP20.0%.txt
10TS_4C_2P200.0%_2AP50.0%.txt

File Name: 10TS_4C_2P200.0%_2AP50.0%.txt

Files of Type: All Files

11:50 AM
10/26/2016

Figure: A.5: Scheduler Interface showing selection of input file



Create New Taskset [✓ Select Existing Taskset]

C:\Users\Dell\Desktop\outputs\10TS_4C_2P200.0%_2AP50.0%.txt

Set Number of Processors: 1 (list: 1, 2, 3, 4, 5, 6, 7, 8; 4 highlighted)

Select Scheduler: MCS Type1 Scheduler

Taskset Partition Scheme: WFD

ution Mode

e Mode Analysis Mode

ed Logger DVFS Logger

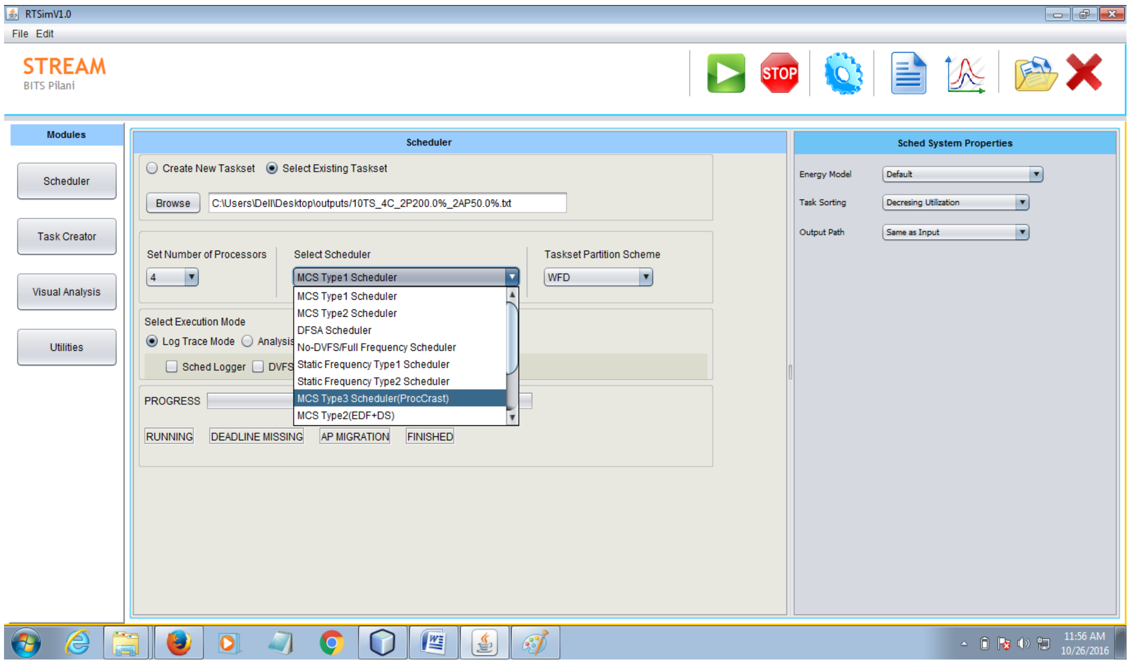
0%

Energy Model: Default

Task Sorting: Decresing Utilization

Output Path: Same as Input

Figure A.6: Scheduler Interface showing selection of processor core count



Create New Taskset [✓ Select Existing Taskset]

C:\Users\Dell\Desktop\outputs\10TS_4C_2P200.0%_2AP50.0%.txt

Set Number of Processors: 4

Select Scheduler: MCS Type1 Scheduler

- MCS Type1 Scheduler
- MCS Type2 Scheduler
- DFSA Scheduler
- No-DVFS/Full Frequency Scheduler
- Static Frequency Type1 Scheduler
- Static Frequency Type2 Scheduler
- MCS Type3 Scheduler(ProcCrast)
- MCS Type2(EDF+DS)

Taskset Partition Scheme: WFD

Select Execution Mode

[✓ Log Trace Mode] Analysis

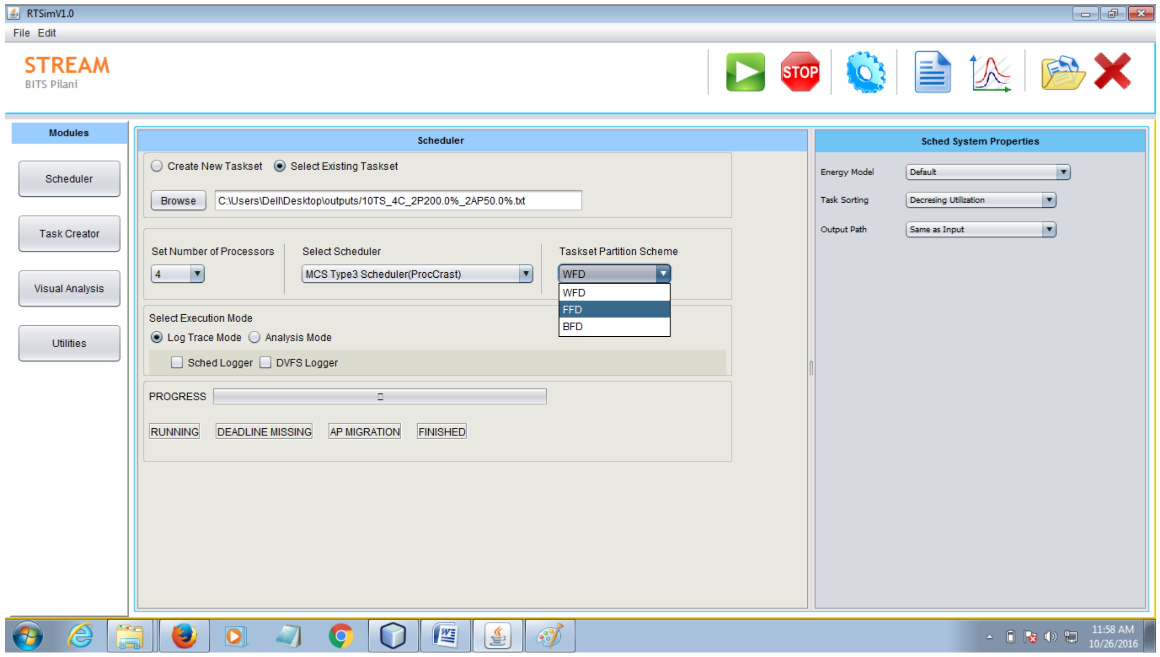
Sched Logger DVFS

Energy Model: Default

Task Sorting: Decresing Utilization

Output Path: Same as Input

Figure A.7: Scheduler Interface showing selection of scheduling Algorithm



Create New Taskset [✓ Select Existing Taskset]

C:\Users\Dell\Desktop\outputs/10TS_4C_2P200.0%_2AP50.0%.txt

Set Number of Processors: 4

Select Scheduler: MCS Type3 Scheduler(ProcCrast)

Taskset Partition Scheme: WFD (WFD, [✓ FFD], BFD)

Select Execution Mode: [✓ Log Trace Mode] Analysis Mode

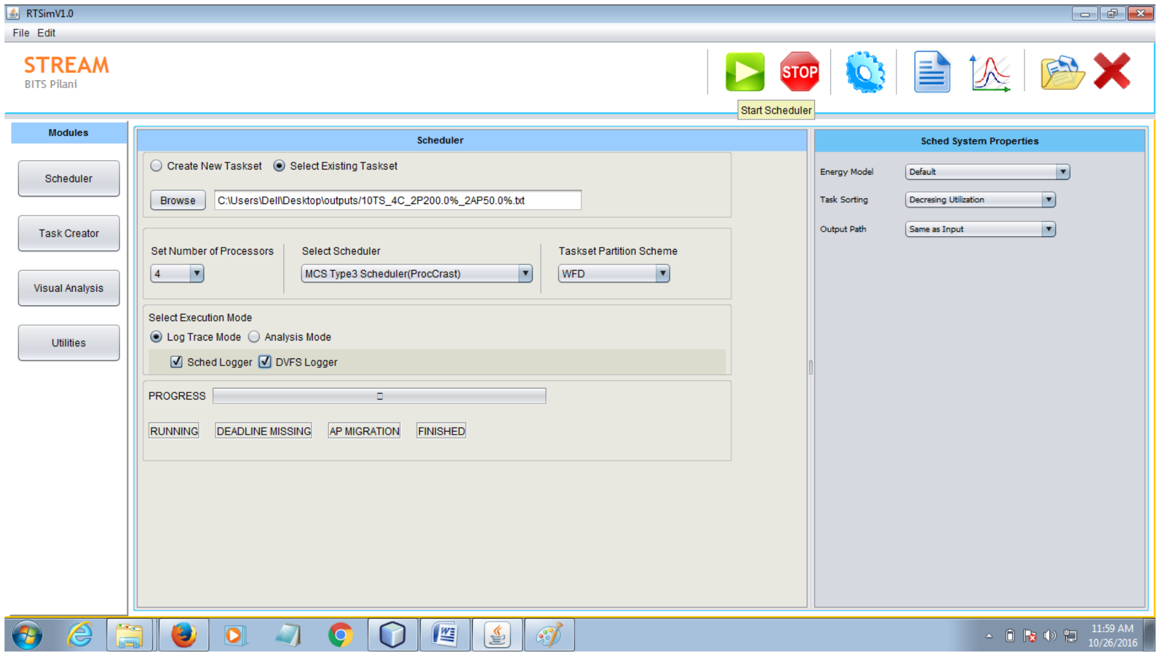
Sched Logger DVFS Logger

Energy Model: Default

Task Sorting: Decresing Utilization

Output Path: Same as Input

Figure A.8: Scheduler Interface showing selection of Task Allocation



Create New Taskset [✓ Select Existing Taskset]

C:\Users\Dell\Desktop\outputs/10TS_4C_2P200.0%_2AP50.0%.txt

Set Number of Processors: 4

Select Scheduler: MCS Type3 Scheduler(ProcCrast)

Taskset Partition Scheme: WFD

Select Execution Mode: [✓ Log Trace Mode] Analysis Mode

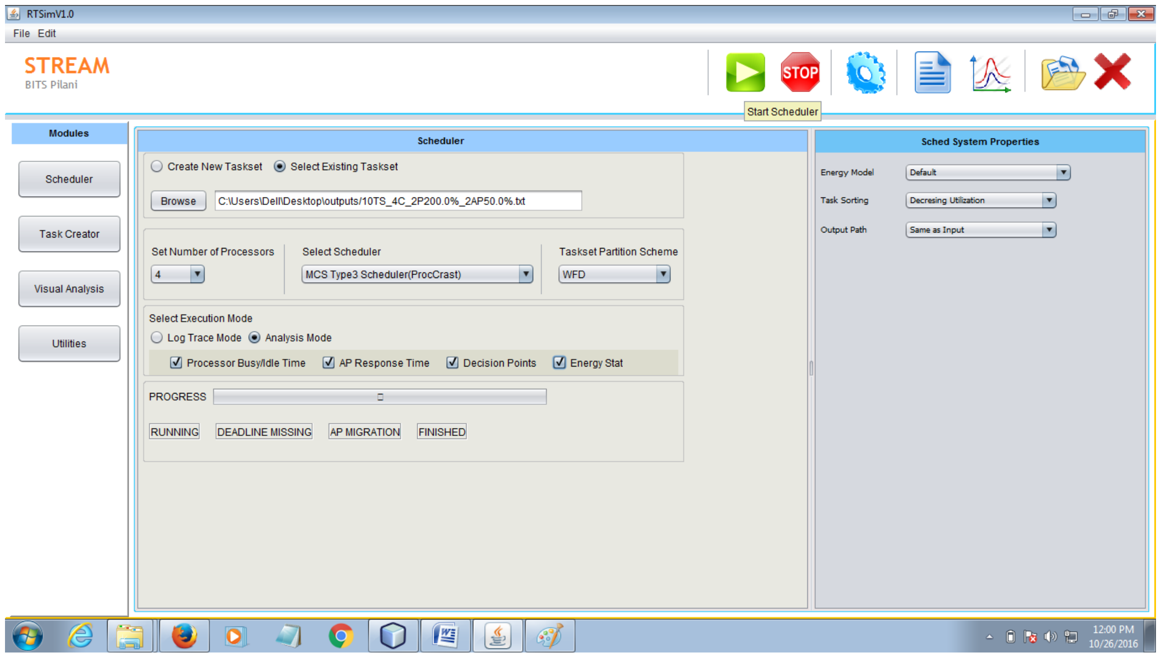
[✓ Sched Logger] [✓ DVFS Logger]

Energy Model: Default

Task Sorting: Decresing Utilization

Output Path: Same as Input

Figure A.9: Scheduler Interface showing selection of Log Trace Mode



Scheduler

Create New Taskset [✓ Select Existing Taskset]

Browse C:\Users\Dell\Desktop\outputs/10TS_4C_2P200.0%_2AP50.0%.txt

Set Number of Processors	Select Scheduler	Taskset Partition Scheme
4	MCS Type3 Scheduler(ProcCrast)	WFD

Select Execution Mode

Log Trace Mode [✓ Analysis Mode]

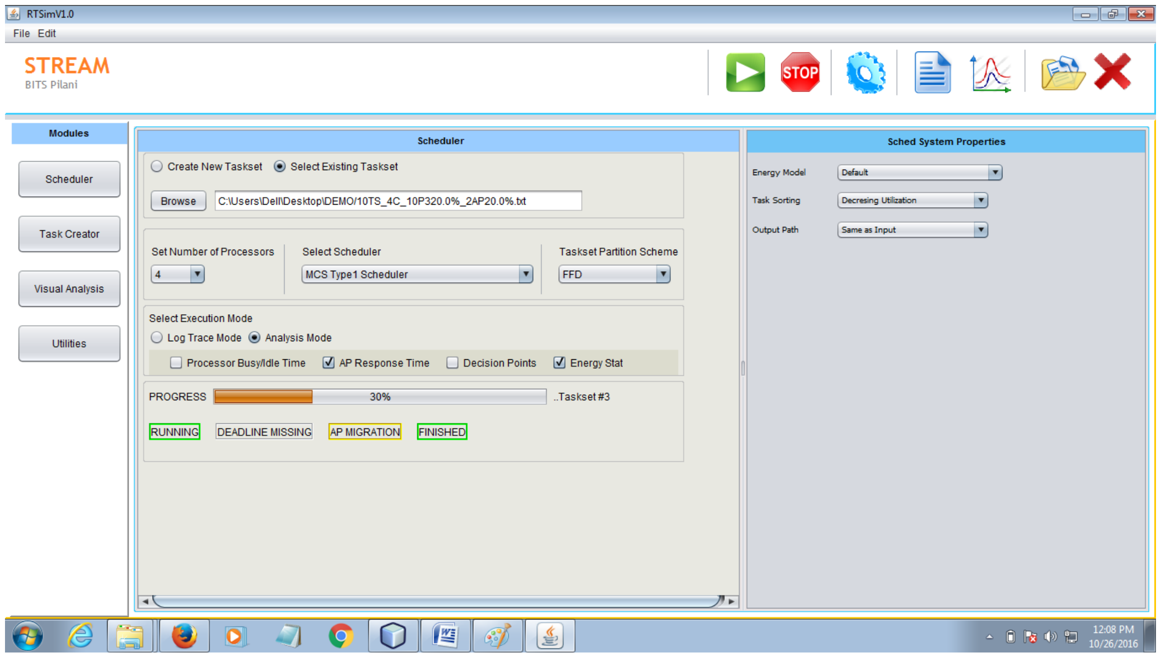
[✓ Processor Busy/Idle Time] [✓ AP Response Time] [✓ Decision Points] [✓ Energy Stat]

PROGRESS

Sched System Properties

Energy Model	Default
Task Sorting	Decresing Utilization
Output Path	Same as Input

Figure A.10: Scheduler Interface showing selection of Analysis Mode



Scheduler

Create New Taskset [✓ Select Existing Taskset]

Browse C:\Users\Dell\Desktop\DEMO/10TS_4C_10P320.0%_2AP20.0%.txt

Set Number of Processors	Select Scheduler	Taskset Partition Scheme
4	MCS Type1 Scheduler	FFD

Select Execution Mode

Log Trace Mode [✓ Analysis Mode]

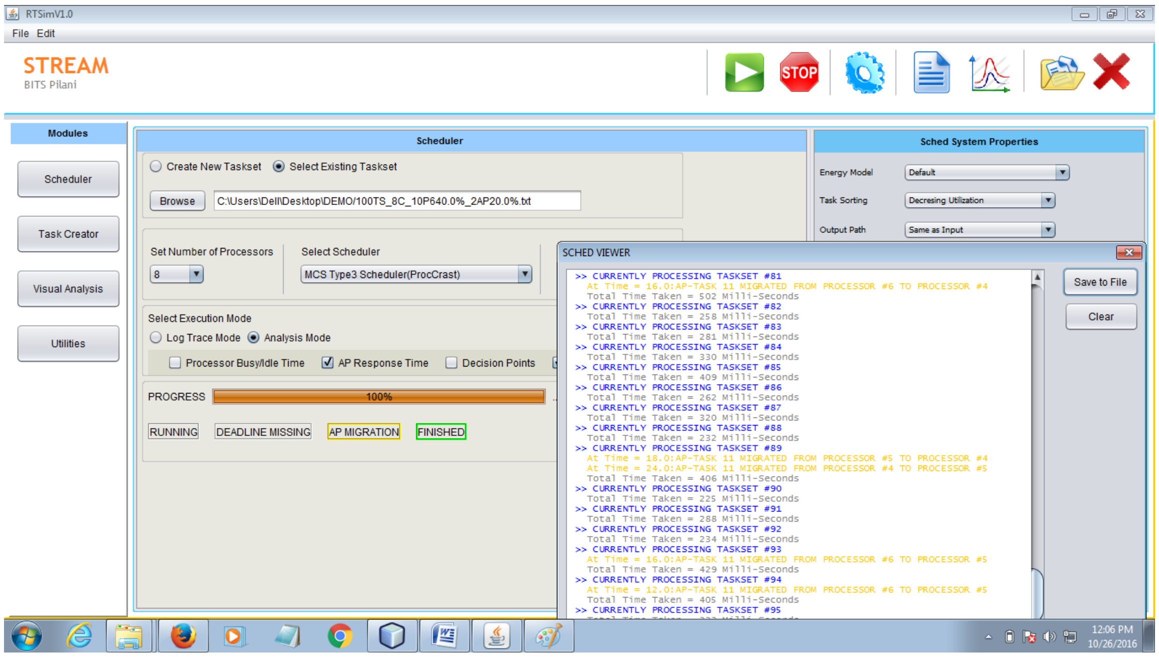
Processor Busy/Idle Time [✓ AP Response Time] Decision Points [✓ Energy Stat]

PROGRESS 30% ..Taskset #3

Sched System Properties

Energy Model	Default
Task Sorting	Decresing Utilization
Output Path	Same as Input

Figure A.11: Scheduler Interface showing Running state of Scheduler



( ) Create New Taskset [✓ Select Existing Taskset]

C:\Users\Dell\Desktop\DEMO\100TS_8C_10P640.0%_2AP20.0%.txt

Set Number of Processors: 8

Select Scheduler: MCS Type3 Scheduler(ProcCrast)

Select Execution Mode: ( ) Log Trace Mode [✓ Analysis Mode]

☐ Processor Busy/Idle Time ☑ AP Response Time ☐ Decision Points

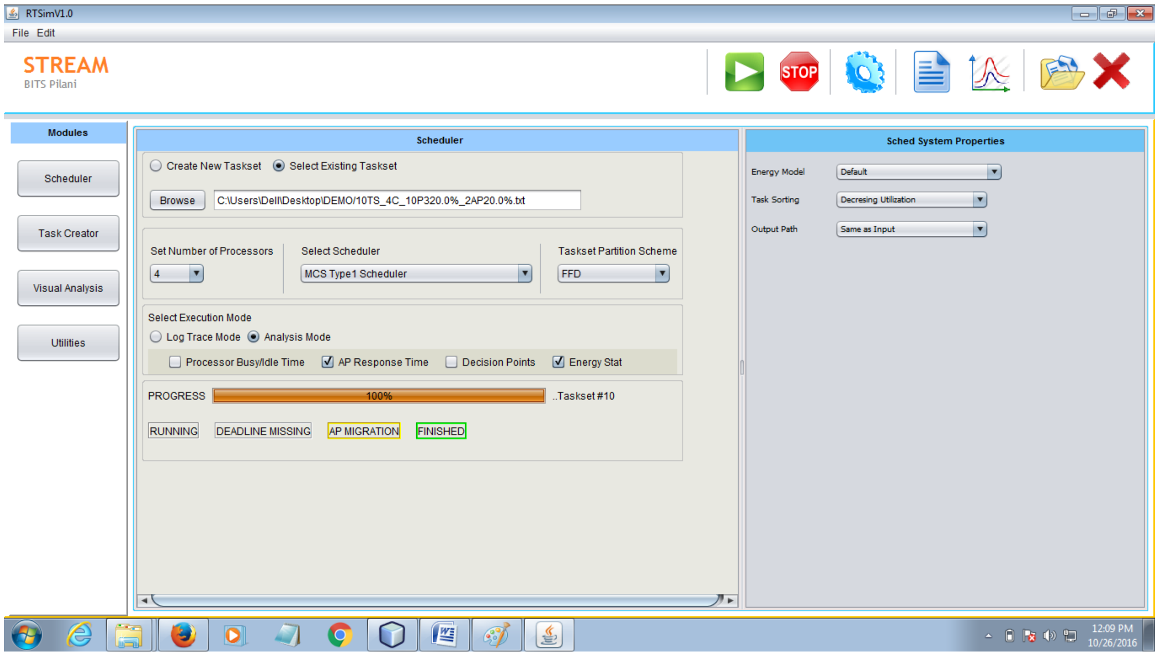
Energy Model: Default

Task Sorting: Decresing Utilization

Output Path: Same as Input

```
>> CURRENTLY PROCESSING TASKSET #81
  At Time = 16.0:AP-TASK 11 MIGRATED FROM PROCESSOR #6 TO PROCESSOR #4
  Total Time Taken = 502 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #82
  Total Time Taken = 258 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #83
  Total Time Taken = 281 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #84
  Total Time Taken = 330 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #85
  Total Time Taken = 409 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #86
  Total Time Taken = 262 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #87
  Total Time Taken = 320 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #88
  Total Time Taken = 232 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #89
  At Time = 18.0:AP-TASK 11 MIGRATED FROM PROCESSOR #5 TO PROCESSOR #4
  At Time = 24.0:AP-TASK 11 MIGRATED FROM PROCESSOR #4 TO PROCESSOR #5
  Total Time Taken = 406 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #90
  Total Time Taken = 225 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #91
  Total Time Taken = 288 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #92
  Total Time Taken = 234 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #93
  At Time = 16.0:AP-TASK 11 MIGRATED FROM PROCESSOR #6 TO PROCESSOR #5
  Total Time Taken = 429 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #94
  At Time = 12.0:AP-TASK 11 MIGRATED FROM PROCESSOR #6 TO PROCESSOR #5
  Total Time Taken = 405 Milli-Seconds
>> CURRENTLY PROCESSING TASKSET #95
```

Figure A.12: Scheduler Interface with SCHED VIEWER



( ) Create New Taskset [✓ Select Existing Taskset]

C:\Users\Dell\Desktop\DEMO\10TS_4C_10P320.0%_2AP20.0%.txt

Set Number of Processors: 4

Select Scheduler: MCS Type1 Scheduler

Taskset Partition Scheme: FFD

Select Execution Mode: ( ) Log Trace Mode [✓ Analysis Mode]

☐ Processor Busy/Idle Time ☑ AP Response Time ☐ Decision Points ☑ Energy Stat

..Taskset #10

Energy Model: Default

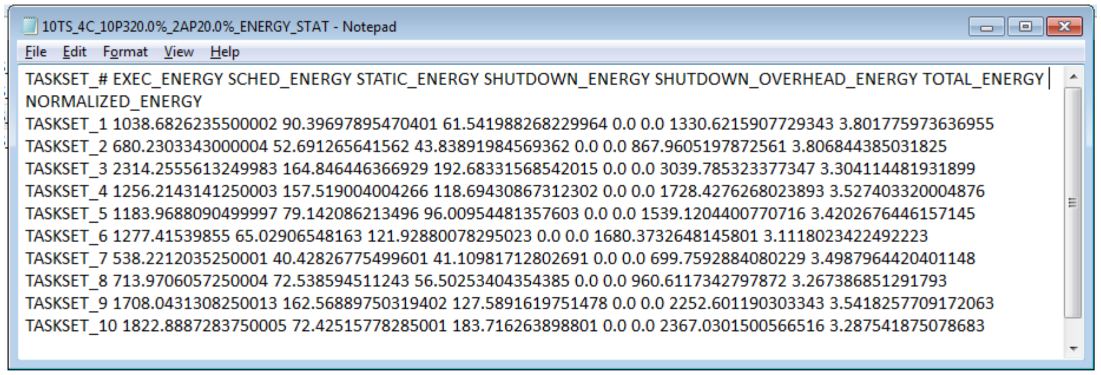
Task Sorting: Decresing Utilization

Output Path: Same as Input

Figure A.13: Scheduler Interface when the execution is finished

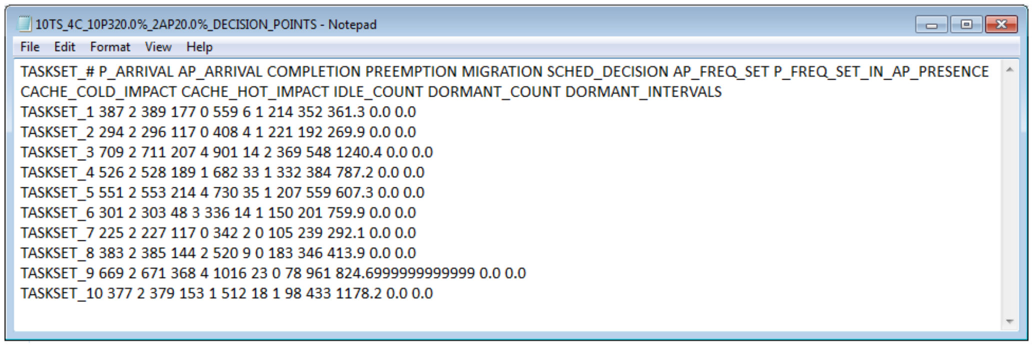
A.4 Output and Performance Analyzer

The output files are produced based on the execution mode specified by the user while running the scheduler. The sample output text files are shown in figures A.14, A.15 and A.16. The output values stored in these files are space separated and can be easily used for further analysis.


10TS_4C_10P320.0%_2AP20.0%_ENERGY_STAT - Notepad
File Edit Format View Help

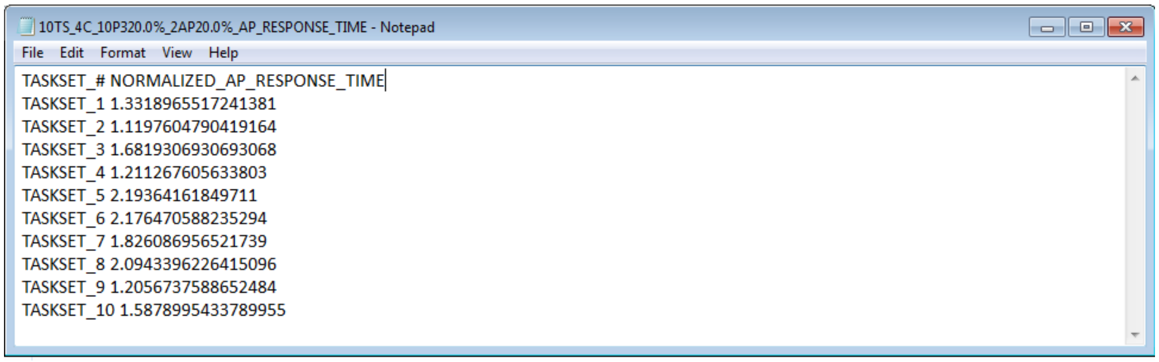
```
TASKSET_# EXEC_ENERGY SCHED_ENERGY STATIC_ENERGY SHUTDOWN_ENERGY SHUTDOWN_OVERHEAD_ENERGY TOTAL_ENERGY
NORMALIZED_ENERGY
TASKSET_1 1038.6826235500002 90.39697895470401 61.541988268229964 0.0 0.0 1330.6215907729343 3.801775973636955
TASKSET_2 680.2303343000004 52.691265641562 43.83891984569362 0.0 0.0 867.9605197872561 3.806844385031825
TASKSET_3 2314.2555613249983 164.846446366929 192.68331568542015 0.0 0.0 3039.785323377347 3.304114481931899
TASKSET_4 1256.2143141250003 157.519004004266 118.69430867312302 0.0 0.0 1728.4276268023893 3.527403320004876
TASKSET_5 1183.9688090499997 79.142086213496 96.00954481357603 0.0 0.0 1539.1204400770716 3.4202676446157145
TASKSET_6 1277.41539855 65.02906548163 121.92880078295023 0.0 0.0 1680.3732648145801 3.1118023422492223
TASKSET_7 538.2212035250001 40.42826775499601 41.10981712802691 0.0 0.0 699.7592884080229 3.4987964420401148
TASKSET_8 713.9706057250004 72.538594511243 56.50253404354385 0.0 0.0 960.6117342797872 3.267386851291793
TASKSET_9 1708.0431308250013 162.56889750319402 127.5891619751478 0.0 0.0 2252.601190303343 3.5418257709172063
TASKSET_10 1822.8887283750005 72.42515778285001 183.716263898801 0.0 0.0 2367.0301500566516 3.287541875078683
```

Figure A.14: Sample output showing Energy Consumption


10TS_4C_10P320.0%_2AP20.0%_DECISION_POINTS - Notepad
File Edit Format View Help

```
TASKSET_# P_ARRIVAL AP_ARRIVAL COMPLETION PREEMPTION MIGRATION SCHED_DECISION AP_FREQ_SET P_FREQ_SET_IN_AP_PRESENCE
CACHE_COLD_IMPACT CACHE_HOT_IMPACT IDLE_COUNT DORMANT_COUNT DORMANT_INTERVALS
TASKSET_1 387 2 389 177 0 559 6 1 214 352 361.3 0.0 0.0
TASKSET_2 294 2 296 117 0 408 4 1 221 192 269.9 0.0 0.0
TASKSET_3 709 2 711 207 4 901 14 2 369 548 1240.4 0.0 0.0
TASKSET_4 526 2 528 189 1 682 33 1 332 384 787.2 0.0 0.0
TASKSET_5 551 2 553 214 4 730 35 1 207 559 607.3 0.0 0.0
TASKSET_6 301 2 303 48 3 336 14 1 150 201 759.9 0.0 0.0
TASKSET_7 225 2 227 117 0 342 2 0 105 239 292.1 0.0 0.0
TASKSET_8 383 2 385 144 2 520 9 0 183 346 413.9 0.0 0.0
TASKSET_9 669 2 671 368 4 1016 23 0 78 961 824.6999999999999 0.0 0.0
TASKSET_10 377 2 379 153 1 512 18 1 98 433 1178.2 0.0 0.0
```

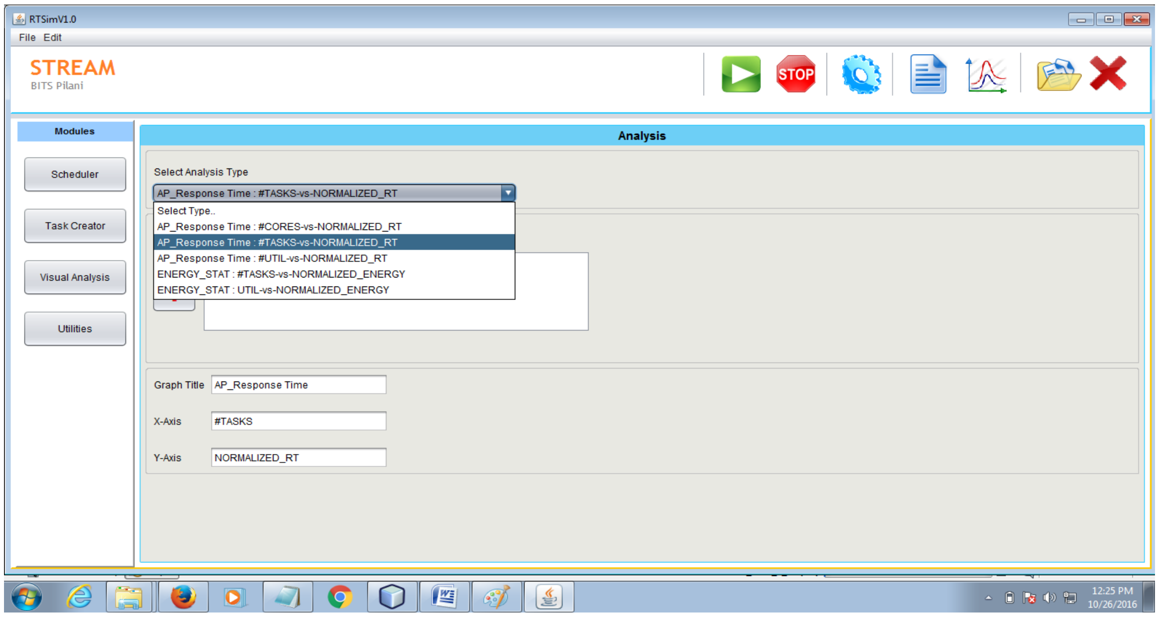
Figure A.15: Sample output showing various Decision Points


10TS_4C_10P320.0%_2AP20.0%_AP_RESPONSE_TIME - Notepad
File Edit Format View Help

```
TASKSET_# NORMALIZED_AP_RESPONSE_TIME
TASKSET_1 1.3318965517241381
TASKSET_2 1.1197604790419164
TASKSET_3 1.6819306930693068
TASKSET_4 1.211267605633803
TASKSET_5 2.19364161849711
TASKSET_6 2.176470588235294
TASKSET_7 1.826086956521739
TASKSET_8 2.0943396226415096
TASKSET_9 1.2056737588652484
TASKSET_10 1.5878995433789955
```

Figure A.16: Sample output showing Aperiodic Response Time

One of the new features of this simulator is the visual plotter tool in visual analysis tab. This feature can be used to plot the graphs for visually analyzing the performance of the scheduling algorithms. The user has to first select the parameters of comparison and then link the output files which are required for plotting the graphs. Then by clicking the graph plotter button, it generates the plot. In addition to the graph, it also generates a text file corresponding to the graph. This tool can be used to test the performance of the proposed scheduling algorithm with the existing scheduling algorithms. Figures A.17, A.18, A.19 and A.20 present the sequence of snapshots for showing the step wise method to use the visual analyzer.

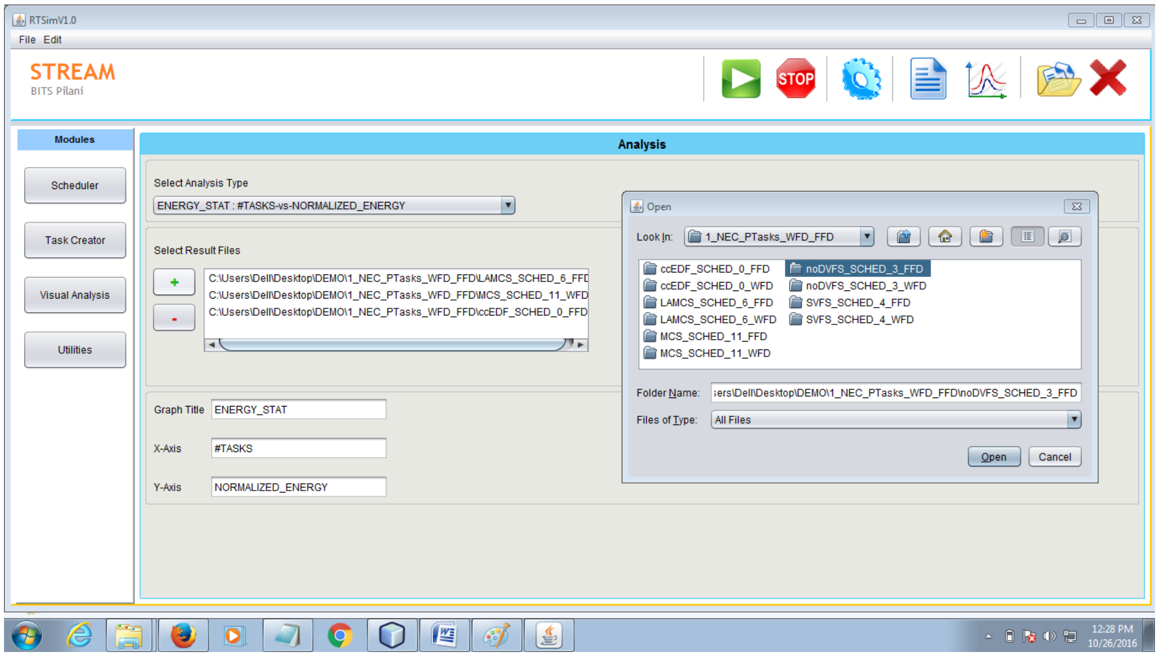


Select Analysis Type
AP_Response Time : #TASKS-vs-NORMALIZED_RT

Select Type..
AP_Response Time : #CORES-vs-NORMALIZED_RT
AP_Response Time : #TASKS-vs-NORMALIZED_RT
AP_Response Time : #UTIL-vs-NORMALIZED_RT
ENERGY_STAT : #TASKS-vs-NORMALIZED_ENERGY
ENERGY_STAT : UTIL-vs-NORMALIZED_ENERGY

Graph Title AP_Response Time
X-Axis #TASKS
Y-Axis NORMALIZED_RT

Figure A.17: Visual Analyzer showing selection of Analysis Type



Select Analysis Type: ENERGY_STAT : #TASKS-vs-NORMALIZED_ENERGY

Select Result Files
C:\Users\Dell\Desktop\DEMO\1_NEC_PTasks_WFD_FFD\LAMCS_SCHED_6_FFD
C:\Users\Dell\Desktop\DEMO\1_NEC_PTasks_WFD_FFD\MCS_SCHED_11_WFD
C:\Users\Dell\Desktop\DEMO\1_NEC_PTasks_WFD_FFD\ccEDF_SCHED_0_FFD

Graph Title: ENERGY_STAT
X-Axis: #TASKS
Y-Axis: NORMALIZED_ENERGY

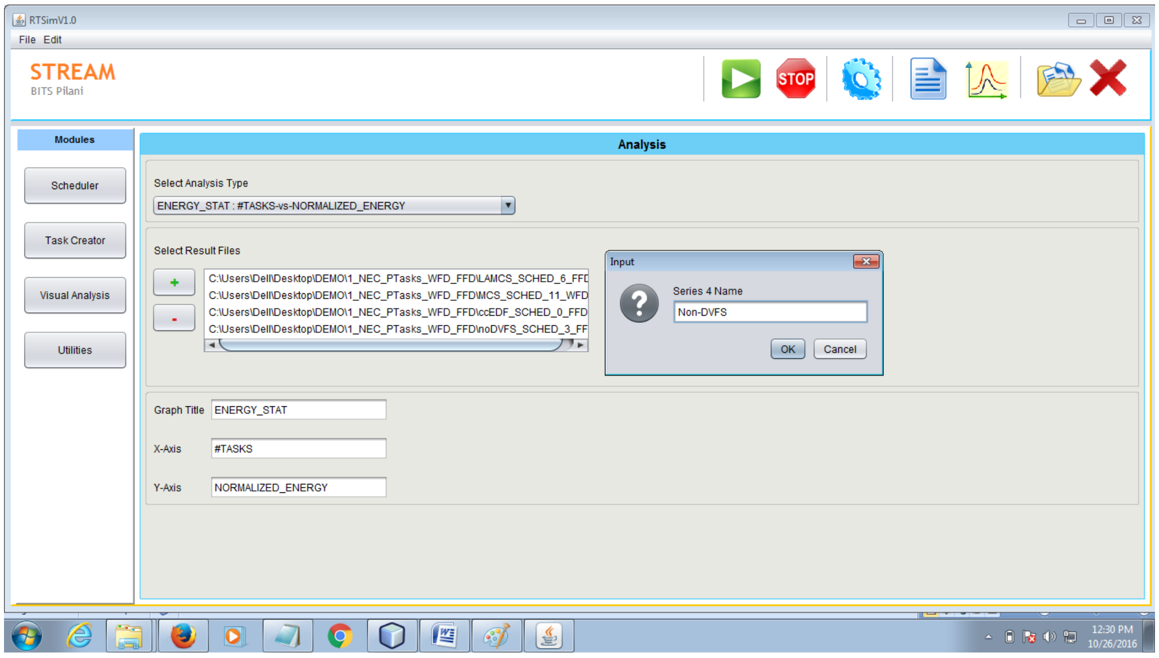
Look In: 1_NEC_PTasks_WFD_FFD

ccEDF_SCHED_0_FFD	noDVFS_SCHED_3_FFD
ccEDF_SCHED_0_WFD	noDVFS_SCHED_3_WFD
LAMCS_SCHED_6_FFD	SVFS_SCHED_4_FFD
LAMCS_SCHED_6_WFD	SVFS_SCHED_4_WFD
MCS_SCHED_11_FFD	
MCS_SCHED_11_WFD	

Folder Name: :ers\Dell\Desktop\DEMO\1_NEC_PTasks_WFD_FFD\noDVFS_SCHED_3_FFD
Files of Type: All Files

12:28 PM
10/26/2016

Figure A.18: Visual Analyzer showing selection of Output Files



Select Analysis Type: ENERGY_STAT : #TASKS-vs-NORMALIZED_ENERGY

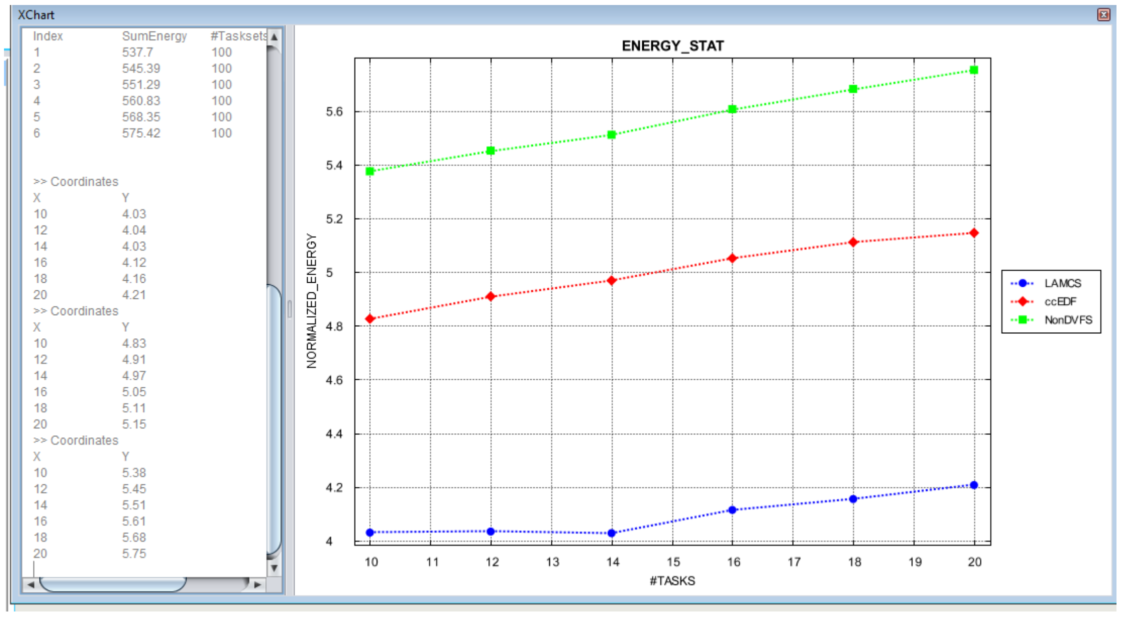
Select Result Files
C:\Users\Dell\Desktop\DEMO\1_NEC_PTasks_WFD_FFD\LAMCS_SCHED_6_FFD
C:\Users\Dell\Desktop\DEMO\1_NEC_PTasks_WFD_FFD\MCS_SCHED_11_WFD
C:\Users\Dell\Desktop\DEMO\1_NEC_PTasks_WFD_FFD\ccEDF_SCHED_0_FFD
C:\Users\Dell\Desktop\DEMO\1_NEC_PTasks_WFD_FFD\noDVFS_SCHED_3_FF

Series 4 Name: Non-DVFS

Graph Title: ENERGY_STAT
X-Axis: #TASKS
Y-Axis: NORMALIZED_ENERGY

12:30 PM
10/26/2016

Figure A.19: Visual Analyzer labeling a line that will appear on the graph



```
XChart
Index        SumEnergy     #Tasksets
1            537.7         100
2            545.39        100
3            551.29        100
4            560.83        100
5            568.35        100
6            575.42        100


>> Coordinates
X            Y
10           4.03
12           4.04
14           4.03
16           4.12
18           4.16
20           4.21
>> Coordinates
X            Y
10           4.83
12           4.91
14           4.97
16           5.05
18           5.11
20           5.15
>> Coordinates
X            Y
10           5.38
12           5.45
14           5.51
16           5.61
18           5.68
20           5.75
```

Figure A.20: Visual Analyzer showing Graph and the Text File corresponding to that graph

References

[1] Ahmed, R., Ramanathan, P., Saluja, K., & Yao, C. (2013). Scheduling aperiodic tasks in next generation embedded real-time systems. *Proceedings of 26th International Conference on VLSI Design and 12th International Conference On Embedded Systems*, pp. 25-30.

[2] Andersson, B. & Jonsson, J. (2003). The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. *Proceeding of 15th IEEE Euromicro Conference on Real Time Systems*, pp. 33-40.

[3] Andersson, B. & Tovar, E. (2006). Multiprocessor scheduling with few preemptions. *Proceedings of 12th IEEE International Conference On Embedded And Real-Time Computing Systems And Applications (RTCSA'06)*, pp. 322-334.

[4] Andersson, B., Abdelzaher, T., & Jonsson, J. (2003a). Global priority-driven aperiodic scheduling on multiprocessors. *Proceedings of International Parallel And Distributed Processing Symposium*, pp. 8.

[5] Andersson, B., Abdelzaher, T., & Jonsson, J. (2003b). Partitioned aperiodic scheduling on multiprocessors. *Proceedings of International Parallel and Distributed Processing Symposium*.

[6] Audsley, N. C., Burns, A., Richardson, M. F. & Wellings, A. J., (1994). STRESS: a simulator for hard real-time systems. *International Journal of Software Practice and Experience*, Vol. 24, No. 6, pp. 543–564.

[7] Awan, M. & Petters, S. (2011). Enhanced race-To-halt: A leakage-Aware energy management approach for dynamic priority systems. *Proceedings of 23rd Euromicro Conference On Real-Time Systems*, pp. 92-101.

[8] Aydin, H. & Yang, Q. (2003). Energy-aware partitioning for multiprocessor real-time systems. *Proceedings of International Parallel And Distributed Processing Symposium*, pp. 9.

[9] Aydin, H. & Yang, Q. (2004). Energy - responsiveness tradeoffs for real-time systems with mixed workload. *Proceedings of 10th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS 2004)*, pp. 74-83.

[10] Aydin, H., Melhem, R., Mosse, D., & Mejia-Alvarez, P. (2001). Dynamic and aggressive scheduling techniques for power-aware real-time systems. *Proceedings of 22nd IEEE Real-Time Systems Symposium (RTSS 2001)* (Cat. No.01PR1420), pp. 95-105.

[11] Aydin, H., Melhem, R., Mosse, D., & Mejia-Alvarez, P. (2004). Power-aware scheduling for periodic real-time tasks. *IEEE Transactions On Computers*, Vol. 53, No. 5, pp. 584-600.

[12] Baruah, S. & Lipari, G. (2004a). A multiprocessor implementation of the total bandwidth server. *Proceedings of 18th International Parallel And Distributed Processing Symposium*, pp. 40.

[13] Baruah, S. & Lipari, G. (2004b). Executing aperiodic jobs in a multiprocessor constant-bandwidth server implementation. *Proceedings of 16th Euromicro Conference On Real-Time Systems (ECRTS 2004)*, pp. 109-116.

[14] Baruah, S. (2013). Partitioned EDF scheduling: a closer look. *International Journal of Real-Time Systems*, Vol. 49, No. 6, pp. 715-729.

[15] Baruah, S., Cohen, N., Plaxton, C., & Varvel, D. (1996). Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, Vol. 15, No. 6, pp. 600-625.

[16] Benini, L., Siegel P., & Micheli, G. D. (1994). Automatic synthesis of gated-clocks for power reduction in sequential circuits. *IEEE Design and Test of Computers, Special Issue on Low Power*, Mag., pp. 32-40.

[17] Bhatti, M., Belleudy, C., & Auguin, M. (2011). Hybrid power management in real time embedded systems: an interplay of DVFS and DPM techniques. *International Journal of Real-Time Systems*, Vol. 47, No. 2, pp. 143-162.

[18] Blumenthal, J., Hildebrandt, J., & Golatowski, F. (2003). YASA-A framework for validation, test, and analysis of real-time scheduling algorithms. *Proceedings of 5th Real-Time Linux Workshop*, pp. 197-204.

[19] Borah, M., Owens, R., & Irwin, M. (1996). Transistor sizing for low power CMOS circuits. *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, Vol. 15, No. 6, pp. 665-671.

[20] Borkar, S. (1999). Design challenges of technology scaling. *IEEE Micro*, Vol. 19, No. 4, pp. 23-29.

[21] Brandenburg, B. & Anderson, J. (2007). Integrating hard/soft real-time tasks and best-effort jobs on multiprocessors. *Proceedings of 19th Euromicro Conference On Real-Time Systems (ECRTS'07)*, pp. 61-70.

[22] Burchard, A., Liebeherr, J., Yingfeng Oh, & Son, S. (1995). New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions On Computers*, Vol. 44, No. 12, pp. 1429-1442.

[23] Burd, T. & Brodersen, R. (1995). Energy efficient CMOS microprocessor design. *Proceedings Of 28th Annual Hawaii International Conference On System Sciences*, Vol.1, pp. 288-297.

[24] Chandarli, Y., Fauberteau, F., Masson, D., Midonnet, S., & Qamhieh, M. (2012). YARTISS: A tool to visualize, test, compare and evaluate real-time scheduling algorithms. *Proceedings of 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, Pisa (Italy), pp. 21-26.

[25] Chandrakasan, A., Sheng, S., & Brodersen, R. (1992). Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 4, pp. 473-484

[26] Chen, G., Huang, K., & Knoll, A. (2014). Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Transactions of Embedded Computing Systems*, Vol. 13, No. 3s, pp. 1-21.

[27] Chen, G., Huang, K., Huang, J., Buckl, C., & Knoll, A. (2013). Effective online power management with adaptive interplay of DVS and DPM for embedded real-

time system. *Proceedings of Euromicro Conference On Digital System Design*, pp. 881-889.

[28] Chen, J. & Kuo, C. (2007a). Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. *Proceedings of 13th IEEE International Conference On Embedded And Real-Time Computing Systems And Applications (RTCSA 2007)*, pp. 28-38.

[29] Chen, J-J & Kuo, T-W. (2007b). Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. *Proceedings of IEEE/ACM International Conference On Computer-Aided Design*, pp. 289-294.

[30] Chen, J. & Kuo, T. (2006). Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. *Proceedings of ACM SIGPLAN/SIGBED Conference on Language, Compilers And Tool Support For Embedded Systems (LCTES '06)*, pp. 153-162.

[31] Chen, J. & Thiele, L. (2008). Expected system energy consumption minimization in leakage-aware DVS systems. *Proceeding of 13th International Symposium On Low Power Electronics And Design (ISLPED '08)*, pp. 315-320.

[32] Chen, J-J., Hsu, H-R., Chuang, K-H., Yang, C-L., Pang, A-C. & Kuo, T-W. (2004). Multiprocessor energy-efficient scheduling with task migration considerations. *Proceedings of 16th Euromicro Conference On Real-Time Systems (ECRTS 2004)*, pp. 101-108.

[33] Chen, Q., Zheng, L., Guo, M., & Huang, Z. (2014). EEWA: Energy-Efficient Workload-Aware Task Scheduling in Multi-core Architectures. *Proceedings of IEEE International Parallel & Distributed Processing Symposium Workshops*, pp. 642-651.

[34] Chen, J-J., Hsu, H-R & Kuo, T-W. (2006). Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. *Proceedings of 12th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS'06)*, pp. 408-417.

[35] Chen, J-J., Yang, C-Y & Kuo, T-W. (2006). Slack Reclamation for Real-Time Task Scheduling over Dynamic Voltage Scaling Multiprocessors. *Proceedings of IEEE International Conference On Sensor Networks, Ubiquitous, And Trustworthy Computing (SUTC'06)*, Vol. 1, pp. 8.

[36] Cho, H., Ravindran, B., & Jensen, E. (2006). An optimal real-time scheduling algorithm for multiprocessors. *Proceedings of 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pp. 101-110.

[37] Cong, J. & Gururaj, K. (2009). Energy efficient multiprocessor task scheduling under input-dependent variation. *Proceedings of Design, Automation & Test In Europe Conference and Exhibition (DATE'09)*, pp. 411-416.

[38] Courbin P. & George, L. (2011). FORTAS: Framework for real-time analysis and simulation. *Proceedings of 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, pp. 21–26.

[39] Davis, R. I. & Burns, A. (2009). Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Proceedings of 30th IEEE Real-Time Systems Symposium (RTSS 2009)*, pp. 398–409.

[40] Davis, R. & Burns, A. (2011). A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Survey*, Vol. 43, No. 4, pp. 1-44.

[41] Dellinger, M., Garyali, P. & Ravindran, B. (2011). ChronOS Linux: a best-effort real-time multiprocessor Linux kernel. *Proceedings of DAC*, pp. 474–479.

[42] Dertouzos, M. L. (1974). Control robotics: The procedural control of physical processes. *Proceedings of the International Federation for Information Processing Working Conference on Data Semantics*. pp. 807–813.

[43] Devadas, V. & Aydin, H. (2010). Coordinated power management of periodic real-time tasks on chip multiprocessors. *Proceedings of IEEE International Conference On Green Computing*, pp. 61-72.

[44] Diaz, A., Batista, R., Castro, O. (2007). Realtss: a real-time scheduling simulator. *Proceedings of 4th International Conference on Electrical and Electronics Engineering*, pp. 165 - 168.

[45] Digalwar, M., Gahukar, P., & Mohan, S. (2014). Design and development of a real time scheduling algorithm for mixed task set on multi-core processors. *Proceedings of 7th IEEE International Conference On Contemporary Computing (IC3 2014)*, pp. 265-269.

[46] Digalwar, M., Gahukar, P., Mohan, S. & Raveendran, B. K. (2015). STREAM: A Simulation Tool for Energy Efficient Real Time Scheduling and Analysis. *Proceedings of International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) in conjunction with 27th Euromicro Conference on Real Time Systems (ECRTS), Lund, Sweden. July 2015.*

[47] Digalwar, M., Mohan, S., & Raveendran, B. (2013). Dynamic voltage and frequency scaling scheduling algorithm for mixed task set. *Proceedings of IEEE 8th International Conference On Industrial And Information Systems (ICIIS 2013)*, pp. 643-648.

[48] Duarte, D., Vijaykrishnan, N., Irwin, M., & Tsai, Y. (2002). Impact of technology scaling and packaging on dynamic voltage scaling techniques. *Proceedings of 15th Annual IEEE International ASIC/SOC Conference*, pp. 244-248.

[49] Emberson, P., Stafford, R., Davis, R. I. (2010). Techniques for the Synthesis of Multiprocessor Tasksets. *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, pp. 6-11.

[50] Frigo, M., Leiserson, C. E., & Randall, K. H. (1998). The implementation of the Cilk-5 multithreaded language. *ACM Sigplan Notices,* Vol. 33, No. 5, pp. 212-223.

[51] Fu, C., Li, M., & Xue, C. J. (2015a). Race to idle or not: Balancing the memory sleep time with DVS for energy minimization. *Design, Automation & Test in Europe Conference & Exhibition (DATE 2015)*, pp. 13-18.

[52] Fu, C., Zhao, Y., Li, M., & Xue, C. J. (2015b). Maximizing common idle time on multi-core processors with shared memory. *Design, Automation & Test in Europe Conference & Exhibition (DATE 2015)*, pp. 900-903.

[53] Fu, X. & Wang, X. (2011). Utilization-controlled task consolidation for power optimization in multi-core real-time systems. *Proceedings of IEEE 17th International Conference On Embedded And Real-Time Computing Systems And Applications, pp.* 73-82.

[54] Fujii, K., Chishiro, H., Matsutani, H., & Yamasaki, N. (2011). Dynamic voltage and frequency scaling for real-time scheduling on a prioritized SMT processor. *Proceedings of IEEE 17th International Conference On Embedded And Real-Time Computing Systems And Applications*, Vol. 2, pp. 9-15.

[55] Garey, M. & Johnson, D. (1979). Computers and Intractability: A guide to the theory of NP-completeness. *W. H. Freeman*, New York, NY.

[56] Gracioli, G., Frohlich, A., Pellizzoni, R., & Fischmeister, S. (2013). Implementation and evaluation of global and partitioned scheduling in a real-time OS. *International Journal of Real-Time Systems*, Vol. 49, No. 6, pp. 669-714.

[57] Gruian, F. (2001). Hard real-time scheduling using stochastic data and DVS processors. *Proceedings of International Symposium on Low Power Electronics and Design*, pp. 46– 51.

[58] Guan, N., Stigge, M., Yi, W., & Yu, G. (2010). Fixed-priority multiprocessor scheduling with Liu and Layland's utilization bound. *Proceedings of 16th IEEE Real-Time And Embedded Technology And Applications Symposium*, pp. 166-174.

[59] Hangan, A., & Sebestyen, G., (2012). RTMultiSim: A versatile simulator for multiprocessor real-time systems. *Proceedings of 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, Pisa (Italy), Vol. 15.

[60] Harbour, M. G., Garcia, J. J. G., Gutierrez, J. C. P., & Drake, J. M., (2001). MAST: Modeling and analysis suite for real time applications. *Proceedings of 13th Euromicro Conference on Real-Time Systems*, pp.125-134.

[61] He, D. & Mueller, W. (2012a). A heuristic energy-aware approach for hard real-time systems on multi-core platforms. *Proceedings of 15th Euromicro Conference On Digital System Design*, pp. 288-295.

[62] He, D. & Mueller, W. (2012b). Enhanced schedulability analysis of hard real-time systems on power manageable multi-core platforms. *Proceedings of IEEE 14th International Conference On High Performance Computing And Communication & IEEE 9th International Conference On Embedded Software And Systems*, pp. 1748-1753.

[63] Hsu, C-H. & Kremer, U. (2003). The Design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. *Proceedings of ACM Conference on PLDI*, pp. 38-48.

[64] Huang, H., Xia, F., Wang, J., Lei, S., & Wu, G. (2010). Leakage-aware reallocation for periodic real-time tasks on multi-core processors. *Proceedings of 5th International Conference On Frontier Of Computer Science And Technology*, pp. 85-91.

[65] Huang, K., Santinelli, L., Chen, J., Thiele, L., & Buttazzo, G. (2009). Adaptive dynamic power management for hard real-time systems. *Proceedings of 30th IEEE Real-Time Systems Symposium*, pp. 23-32.

[66] Isci, C., Buyuktosunoglu, A., Cher, C. Y., Bose, P., & Martonosi, M. (2006). An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget. *Proceedings of 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pp. 347-358.

[67] Islam, F. & Lin, M. (2014). Learning based power management for periodic real-time tasks. Proceedings of IEEE International Conference On High Performance Computing And Communications, *IEEE 6th International Symposium On Cyberspace Safety And Security, IEEE 11th International Conference On Embedded Software And Systems (HPCC, CSS, ICESS)*, pp. 534-541.

[68] Islam, F. & Lin, M. (2015). A framework for learning based DVFS technique selection and frequency scaling for multi-core real-time systems. *Proceedings of IEEE 17th International Conference On High Performance Computing And Communications, IEEE 7th International Symposium On Cyberspace Safety and*

Security and IEEE 12th International Conference On Embedded Software And Systems, pp. 721-726.

[69] Jakovljevic, G., Rakamaric, Z., & Babic, D. (2002). Java simulator of real-time scheduling algorithms. *Proceedings of 24th International Conference on Information Technology Interfaces*, Cavtat, Croatia, pp. 411-416.

[70] Jejurikar, R. & Gupta, R. (2004). Dynamic voltage scaling for system wide energy minimization in real-time embedded systems. *Proceedings of International Symposium On Low Power Electronics And Design (ISLPED '04)*, pp. 76-81.

[71] Jejurikar, R. & Gupta, R. (2005). Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. *Proceedings of 42nd Annual Design Automation Conference - DAC '05*, pp. 111-116.

[72] Jejurikar, R., Pereira, C., & Gupta, R. (2004). Leakage aware dynamic voltage scaling for real-time embedded systems. *Proceedings of 41st Annual Design Automation Conference - DAC '04*, pp. 275-280.

[73] Jha, N. (2005). Low-power system scheduling, synthesis and displays. *Proceedings of IEE Computers And Digital Techniques*, Vol. 152, No. 3, pp. 344-352.

[74] Jin, S., Pei, S., Han, Y., & Li, H. (2015). On optimizing system energy of multi-core SoCs based on dynamically reconfigurable voltage-frequency island. *Proceedings of IEEE Conference on VLSI Design, Automation And Test (VLSI-DAT)*, pp. 1-4.

[75] Kandhalu, A., Kim, J., Lakshmanan K., Rajkumar, R. (2011). Energy-aware partitioned fixed-priority scheduling for chip multi-processors. *Proceedings of IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications, Toyama*, pp. 93-102.

[76] Kappiah, N., Lowenthal, D.K. & Freeh, V.W. (2006). Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs. Proceedings of ACM/IEEE Supercomputing Conference, pp. 33.

[77] Kato, S. & Yamasaki, N. (2008). Scheduling aperiodic tasks using total bandwidth server on multiprocessors. *Proceedings of IEEE/IFIP International Conference On Embedded And Ubiquitous Computing,* Vol.1, pp. 82-89.

[78] Khalib, Z., Ahmad, B., & Bi, O. (2012). Performance analysis of a non-preemptive dynamic soft real time scheduler using discrete event simulator. *Proceedings of 4th International Conference on Computational Intelligence, Modeling and Simulation*, pp.182-187.

[79] Kim, W., Shin, D., Yun, H-S., Kim, J. & Min, S. L. (2002). Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. *Proceedings of 8th IEEE Real-Time and Embedded Technology And Applications Symposium*, pp. 219-228.

[80] Krishna, C. M. and Shin, K. G. (1997). Real-Time Systems. *Co-published by MIT Press and McGraw-Hill, Inc*.

[81] Kuo, C. (2013). Ratio-based aggressive reclaim algorithm for mixed task sets. *Proceedings of IEEE 10th International Conference On High Performance Computing And Communications and IEEE International Conference On Embedded And Ubiquitous Computing*, pp. 2042-2049.

[82] Kuo, C. (2014). Energy-efficient scheduling for real-time tasks on uniform multiprocessors. *Proceedings of IEEE 12th International Conference On Dependable, Autonomic And Secure Computing*, pp. 192-195.

[83] Lakshmanan, K., Rajkumar, R., & Lehoczky, J. (2009). Partitioned fixed-priority preemptive scheduling for multi-core processors. *Proceedings of 21st Euromicro Conference On Real-Time Systems*, pp. 239-248.

[84] Langen, P. & Juurlink, B. (2006). Leakage-aware multiprocessor scheduling for low power. *Proceedings of 20th IEEE International Parallel & Distributed Processing Symposium*, pp. 8.

[85] Langen, P. & Juurlink, B. (2009). Leakage-aware multiprocessor scheduling. *Proceedings of Journal of Signal Processing Systems*, Vol. 57, Issue 1, pp 73 - 88.

[86] Lee, C-H. & Shin, K. (2004). On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm. *Proceedings of 25th IEEE International Real-Time Systems Symposium*, pp. 319-335.

[87] Lee, J., Yun, B. & Shin, K. (2014). Reducing peak power consumption in multi-core systems without violating real-time constraints. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 4, pp. 1024-1033.

[88] Lee, S. & Sakurai, T. (2000). Run-time voltage hopping for low power real-time systems. *Proceedings of 37th Design Automation Conference*, pp. 806–809.

[89] Lee, W. (2012). Energy-efficient scheduling of periodic real-time tasks on lightly loaded multi-core processors. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 23, No. 3, pp. 530-537.

[90] Lee, W. Y. (2009). Energy-saving DVFS scheduling of multiple periodic real-time tasks on multi-core processors. *Proceedings of 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, (DS-RT '09),* pp. 216-223.

[91] Lee, W. Y., Ko, Y. W., Lee, H., Kim H. (2009). Energy-efficient scheduling of a real-time task on DVFS-enabled multi-cores. *Proceedings of the International Conference on Hybrid Information Technology*, pp. 273-277.

[92] Lee, Y-H., Reddy, K., & Krishna, C. (2003). Scheduling techniques for reducing leakage power in hard real-time systems. *Proceedings of 15th Euromicro Conference On Real-Time Systems*, pp. 105-112.

[93] Legout, V., Jan, M., & Pautet, L. (2014). Scheduling algorithms to reduce the static energy consumption of real-time systems. *International Journal of Real-Time Systems*, Vol. 51, No. 2, pp. 153-191.

[94] Leung, J. Y.-T. & Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*. Vol. 2, No. 4, pp. 237–250.

[95] Li, D. & Wu, J. (2014). Energy-aware scheduling for aperiodic tasks on multi-core processors. *Proceedings of 43rd International Conference On Parallel Processing*, pp. 361-370.

[96] Li, L., Choi, K., & Nan, H. (2011). Effective algorithm for integrating clock gating and power gating to reduce dynamic and active leakage power simultaneously. *Proceedings of 12th International Symposium On Quality Electronic Design*, pp. 1-6.

[97] Lin, C., Chang, C., Syu, Y., Wu, J., Liu, P., Cheng, P., & Hsu, W. (2014). An energy-efficient task scheduler for multi-core platforms with per-core DVFS based on task characteristics. *Proceedings of 43rd International Conference On Parallel Processing*, pp. 381-390.

[98] Lin, C., Wang, B., & Hsiung, P. (2012). Synchronization-aware dynamic thread scheduling for improving performance and saving energy in multi-core embedded systems. *5th International Symposium On Parallel Architectures, Algorithms And Programming*, pp. 13 - 18.

[99] Liu, C. L. & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, Vol. 20, No. 1, pp. 46–61.

[100] Liu, J. & Guo, J. (2014). Voltage Island Aware Energy Efficient Scheduling of Real-Time Tasks on Multi-core Processors. *Proceedings of IEEE International Conference On High Performance Computing And Communications, IEEE 6th International Symposium On Cyberspace Safety And Security, IEEE 11th International Conference On Embedded Software And Systems (HPCC, CSS, ICESS)*, pp. 645-652.

[101] Liu, Jane. W. S., (2008). Real Time Systems, *Pearson Education, Inc and Dorling Kindersley Publishing Inc*.

[102] Lopez, J., Diaz, J., & Garcia, D. (2004). Utilization bounds for EDF scheduling on real-time multiprocessor systems. *International Journal of Real-Time Systems*, Vol. 28, No. 1, pp. 39-68.

[103] Lorch, J. & Smith, A. (2004). PACE: a new approach to dynamic voltage scaling. *IEEE Transactions On Computers*, Vol. 53, No. 7, pp. 856-869.

[104] Lu, J. & Guo, Y. (2011). Energy-aware fixed-priority multi-core scheduling for real-time systems. *Proceedings of IEEE 17th International Conference On Embedded And Real-Time Computing Systems and Applications* pp. 277-281.

[105] Lundberg, L. & Lennerstad, H. (2008). Slack-based global multiprocessor scheduling of aperiodic tasks in parallel embedded real-time systems. *Proceedings of IEEE/ACS International Conference On Computer Systems And Applications*, pp. 465-472.

[106] Macii, E., Bolzani, L., Calimera, A., Macii, A., & Poncino, M. (2008). Integrating clock gating and power gating for combined dynamic and leakage power optimization in digital CMOS circuits. *Proceedings of 11th EUROMICRO Conference On Digital System Design Architectures, Methods And Tools*, pp. 298-303.

[107] Mall, Rajib. (2007). Real Time Systems - Theory and Practice, *Pearson Education, Dorling Kindersley (India) Pvt. Ltd*.

[108] Manacero, A., Miola, M. B, Nabuco, V. A., (2001). Teaching real-time with a scheduler simulator. *Proceedings of 31st Annual Conference on Frontiers in Education*, Vol.2, pp. T4D - 15.

[109] Martin, S., Flautner, K., Mudge, T., & Blaauw, D. (2002). Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. *Proceedings of IEEE/ACM International Conference On Computer Aided Design (ICCAD 2002)*, pp. 721-725.

[110] Min-Allah, N., Kazmi, A., Ali, I., Jian-Sheng, X., & Yong-Ji, W. (2008). Minimizing response time implication in DVS scheduling for low power embedded systems. *Proceedings of IEEE Conference on Innovations In Information Technologies (IIT)*, pp. 347-351.

[111] Moreno, G. & Niz, D. (2012). An optimal real-time voltage and frequency scaling for uniform multiprocessors. *Proceedings of IEEE International Conference On Embedded And Real-Time Computing Systems And Applications*, pp. 21 - 30.

[112] Nikolic, B., Awan, M.A., & Petters, S.M. (2011). SPARTS: Simulator for power aware and Real-Time Systems. *Proceedings of 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 999 - 1004.

[113] Niu, L. & Quan, G. (2004). Reducing both dynamic and leakage energy consumption for hard real-time systems. *Proceedings of International Conference On Compilers, Architecture, And Synthesis For Embedded Systems (CASES '04)*, pp. 140 - 148.

[114] Niu, L. (2010). Energy Efficient Scheduling for Real-Time Embedded Systems with QoS Guarantee. *Proceedings of IEEE 16th International Conference On Embedded And Real-Time Computing Systems And Applications*, pp. 75-87.

[115] Niu, L. and Quan, G. (2015). Peripheral-conscious energy-efficient scheduling for weakly hard real-time systems. *International Journal of Embedded Systems*, Vol. 7, No. 1, pp. 11–25.

[116] Pagani, S. & Chen, J. (2013). Energy efficient task partitioning based on the single frequency approximation scheme. *Proceedings of IEEE 34th Real-Time Systems Symposium*, pp. 308-318.

[117] Pillai, A.S., & Isha, T.B. (2013). ERTSim: An embedded real-time task simulator for scheduling. *Proceedings of IEEE International Conference on Computational Intelligence and Computing Research*, pp. 1-4.

[118] Pillai, P. & Shin, K. (2001). Real-time dynamic voltage scaling for low-power embedded operating systems. *ACM Operating Systems Review (SIGOPS)*, Vol. 35, No. 5, pp. 89.

[119] Raja, T., Agrawal, V., Bushnell, M. (2006). Transistor sizing of logic gates to maximize input delay variability. *Journal of Low Power Electronics*, Vol. 2, No. 1, pp. 121–128.

[120] Ramirez, A. D., Orduno, D. K., Alvarez, P. M. (2012). A multiprocessor real-time scheduling simulation tool. *Proceedings of 22nd International Conference on Electrical Communications and Computers*, pp. 157-161.

[121] Ren, D. & Suda, R. (2009). Power efficient large matrices multiplication by load scheduling on multi-core and GPU platform with CUDA. *Proceedings of International Conference on Computational Science and Engineering (CSE '09)*, pp. 424-429.

[122] Rivas, J. M., Gutierrez, J. J., Harbour, M. G. (2014). GEN4MAST: A tool for the evaluation of real-time techniques using a supercomputer. *Proceedings of 3rd International Workshop on Real Time and Distributed Computing in Emerging Applications co-located with 34th IEEE Real Time Systems Symposium*.

[123] Roy, K., Mukhopadhyay, S., & Mahmoodi-Meimand, H. (2003). Leakage current mechanisms and leakage reduction techniques in deep-sub micrometer CMOS circuits. *Proceedings of the IEEE*, Vol. 91, No. 2, pp. 305-327.

[124] Saha, S & Raveendran, B. (2012). An experimental evaluation of real-time DVFS scheduling algorithms. *Proceedings of the 5th Annual International Systems and Storage Conference*, pp. 7-19.

[125] Sasaki, H., Buyuktosunoglu, A., Vega, A., & Bose, P. (2016). Mitigating power contention: A scheduling based approach. *IEEE Computer Architecture Letters*, pp. 1-1.

[126] Schirmeister Frank, (2007). Multi-core processors: fundamentals, trends and challenges. *Proceedings of Embedded Systems Conference (ESC)*, Imperas, Inc.

[127] Schoenherr, J. H., Richling, J., Muehl, G., Werner, M. (2010). A scheduling approach for efficient utilization of hardware-driven frequency scaling. *Proceedings of 23rd International Conference on Architecture of Computing Systems (ARCS)*, pp. 1-10.

[128] Scordino, C. & Lipari, G. (2006). A resource reservation algorithm for power-aware scheduling of periodic and aperiodic real-time tasks. *IEEE Transactions On Computers*, Vol. 55, No. 12, pp. 1509-1522.

[129] Sensini, Fabrizio, Buttazzo, G., and Ancilotti, P., (1997). Ghost: A tool for simulation and analysis of real-time scheduling algorithms. *Proceedings of the IEEE Real-Time Educational Workshop*, pp. 42-49.

[130] Seo, E., Jeong, J., Park, S & Lee, J. (2008). Energy efficient scheduling of real-time tasks on multi-core processors. *IEEE Transactions on Parallel and Distributed Systems*. Vol. 19, No. 11, pp. 1540-1552.

[131] Sheikh, H. & Ahmad, I. (2014). Efficient heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-core processors. *Proceedings of International Green Computing Conference*, pp. 1-8.

[132] Sheikh, H. & Ahmad, I. (2015). Niched evolutionary techniques for performance, energy, and temperature optimized scheduling in multi-core systems. *Proceedings of 6th International Green and Sustainable Computing Conference (IGSC 2015)*, pp. 1-6.

[133] Sheikh, H. F. & Ahmad, I. (2012). Simultaneous optimization of performance, energy and temperature for DAG scheduling in multi-core processors. *Proceedings of International Green Computing Conference (IGCC 2012)*, pp. 1-6.

[134] Sheikh, H., Ahmad, I., & Fan, D. (2016). An evolutionary technique for performance energy temperature optimized scheduling of parallel tasks on multi-core processors. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, No. 3, pp. 668-681.

[135] Sheikh, H., Tan, H., Ahmad, I., Ranka, S., & Bv, P. (2012). Energy and performance aware scheduling of tasks on parallel and distributed systems. *ACM Journal on Emerging Technologies in Computing Systems*, Vol. 8, No. 4, pp. 1-37.

[136] Shieh, W. & Chen, B. (2010). Energy-efficient tasks scheduling algorithm for dual-core real-time systems. *IEEE International Computer Symposium (ICS 2010)*, pp. 568-575.

[137] Shin, D. & Kim, J. (2006). Dynamic voltage scaling of mixed task sets in priority-driven systems. *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, Vol. 25, No. 3, pp. 438-453.

[138] Shin, D. & Kim, J. (2004). Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems. *Proceedings of Asia And South Pacific Design Automation Conference (ASP-DAC 2004)*, pp. 653-658.

[139] Shin, D., Kim, J. & Lee, S. (2001). Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. *IEEE Design and Test of Computers*, Vol. 18, No. 2, pp. 20–30.

[140] Shin, D., Kim, J. (2005). Intra-task voltage scheduling on DVS-enabled hard real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 24, No. 10, pp. 1530-1549.

[141] Shin, D., Kim, W., Jeon, J., Kim, J., Min, S. L. (2003). SimDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage and Frequency Scaling Algorithms. *International Workshop on Power-Aware Computer Systems* (PACS 2003), pp. 141–156.

[142] Shin, Y., Choi, K. & Sakurai, T. (2000). Power optimization of real-time embedded systems on variable speed processors. *Proceedings of IEEE/ACM International Conference On Computer Aided Design. (ICCAD 2000)*, pp. 365-368.

[143] Singh, D. & Kaiser, W. (2014). Energy efficient task scheduling on a multi-core platform using real-time energy measurements. *Proceedings of International Symposium On Low Power Electronics And Design (ISLPED '14)*.

[144] Singhoff, F., Legrand, J., Nana, L. and Marce, L. (2004). Cheddar: a flexible real time scheduling framework. *Proceedings of SIGAda*, pp. 1-8.

[145] Sousa, P., Andersson, B., & Tovar, E. (2011). Implementing slot-based task-splitting multiprocessor scheduling. *Proceedings of 6th IEEE International Symposium On Industrial And Embedded Systems*, pp. 256-265.

[146] Springer, R., Lowenthal, D.K., Rountree, B. & Freeh, V.W. (2006). Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster. Proceedings of ACM SIGPLAN Principles and Practice of Parallel Programming (PPoPP '06), pp. 230-238.

[147] Spuri, M. & Buttazzo, G. (1996). Scheduling aperiodic tasks in dynamic priority systems. *International Journal of Real Time Systems*, Vol. 10. No. 2, pp.179–210.

[148] Stalling, W. (2014). Computer Organization and Architecture, Designing for Peroformance. *Pearson Education, Dorling Kindersley.*

[149] Sucha, P., Kutil, M., Sojka, M., & Hanzalek, Z. (2006). Torsche scheduling toolbox for matlab. *Proceedings of IEEE International Conference on Control Applications, IEEE International Symposium on Intelligent Control*, pp. 1181-1186.

[150] Tang, H., Ramanathan, P., & Compton, K. (2011). Combining hard periodic and soft aperiodic real-time task scheduling on heterogeneous compute resources. *Proceedings of International Conference On Parallel Processing*, pp. 753-762.

[151] Tiwari, V., Malik, S., Wolfe, A. & Lee, M. T-C. (1996). Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing*, pp-1-18.

[152] Trans, X.T, Do,T. V. & Chakka R. (2016). The impact of dynamic power management in computational clusters with multi-core processors. *Journal of Scientific and Industrial Research*, Vol. 75, pp. 339-343.

[153] Urunuela, R., Deplanche, A.-M., Trinquet, Y. (2010). STORM: a simulation tool for real-time multiprocessor scheduling evaluation. *Proceedings of IEEE Conference on Emerging Technologies and Factory Automation (ETFA),* pp. 1-8.

[154] Viredaz, M. & Wallach, D. (2003). Power evaluation of a handheld computer. *IEEE Micro*, Vol. 23, No. 1, pp. 66-74.

[155] Vroey, S. D., Goossens, J., Hernalsteen, C., (1996). A generic simulator of real-time scheduling algorithms. *Proceedings of 29th Annual Simulation Symposium*, pp. 242-249, 8-11.

[156] Wei, Y., Yang, C., Kuo, T., Hung, S., & Chu, Y. (2010). Energy-efficient real-time scheduling of multimedia tasks on multi-core processors. *Proceedings of ACM Symposium On Applied Computing (SAC '10)*, pp. 258-262.

[157] Weiser, M., Welch, B., Demers, M. and Shenker, B. (1994). Scheduling for reduced CPU energy. *Proceedings of 1st Symposium on Operating Systems Design and Implementation*, Monterey, CA, November, pp. 13–23.

[158] Wu, H., Ravindran, B., Jensen E. D., Li, P. (2004). CPU scheduling for statistically-assured real-time performance and improved energy efficiency. *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis, CODES*, pp. 110-115.

[159] Wu, J. and Wu, J-X. (2014). An SRP-based energy-efficient scheduling algorithm for dependent real-time tasks. *Proceedings of International Journal of Embedded Systems*, Vol. 6, No. 4, pp. 335–350.

[160] Xian, C., Lu, Y.H. and Li, Z. (2007). Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. *Proceedings of 44th ACM/IEEE Design Automation Conference*, pp. 664–669.

[161] Xu, C., Xue, C., Yi He, & Sha, E. (2010). Energy efficient joint scheduling and multi-core interconnect design. *Proceedings of 15th Asia And South Pacific Design Automation Conference (ASP-DAC)*, pp. 879-884.

[162] Yang, C-Y., Chen, J-J. & Kuo, T-W. (2005). An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. *Proceedings of Conference on Design, Automation And Test In Europe (DATE 2005)*, pp. 468-473.

[163] Younis, S., & Knight, T. (1993). Practical implementation of charge recovering asymptotically zero power CMOS. *Proceedings of Symposium on Integrated Systems, Univ. of Washington*, pp. 234-250.

[164] Yu, H., Veeravalli, B., Ha, Y. (2010). Leakage-aware dynamic scheduling for real-time adaptive applications on multiprocessor systems. *Proceedings of 47th ACM/IEEE Design Automation Conference (DAC)*, pp. 493-498.

[165] Zapata, O.U.P. and Alvarez, P.M. (2004). EDF and RM multiprocessor scheduling algorithms: survey and performance evaluation. Report No. CINVESTAV-CS-RTG-02, CINVESTAV-IPN, Mexico.

[166] Zeng, G., Yokoyama, T., Tomiyama, H., & Takada, H. (2009). Practical energy-aware scheduling for real-time multiprocessor systems. *Proceedings of 15th IEEE International Conference On Embedded And Real-Time Computing Systems And Applications*, pp. 383-392.

[167] Zhang, Z. & Chang, J. (2014). A cool scheduler for multi-core systems exploiting program phases. *IEEE Transactions On Computers*, Vol. 63, No. 5, pp.1061-1073.

[168] Zhao, Y., Li, X., Jia, Z., Ju, L., & Zong, Z. (2013). Dependency-based energy-efficient scheduling for homogeneous multi-core clusters. *Proceedings of 12th IEEE International Conference On Trust, Security And Privacy In Computing And Communications*, pp. 1299-1306.

[169] Zhu, D., Melhem, R. & Childers, B.R. (2003). Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, No. 7, pp.686–700.

List of Publications

1. Digalwar, M., Gahukar, P., Raveendran, B. K. & Mohan, S. (2014). Energy efficient real-time scheduling algorithm for mixed task set on multi-core processors. *International Journal of Embedded Systems, (In Press)*. **(SCOPUS Indexed)**
2. Digalwar, M., Gahukar, P., Mohan, S. & Raveendran, B. K. (2015). STREAM: A Simulation Tool for Energy Efficient Real Time Scheduling and Analysis. *Proceedings of International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) in conjunction with 27th Euromicro Conference on Real Time Systems (ECRTS), Lund, Sweden. July 2015.* (**Tier I Conference**)
3. Digalwar, M., Gahukar, P., & Mohan, S. (2014). Design and development of a real time scheduling algorithm for mixed task set on multi-core processors. *Proceedings of 7th IEEE International Conference on Contemporary Computing (IC3 2014)*, pp. 265-269. **(SCOPUS Indexed)**
4. Digalwar, M., Mohan, S., & Raveendran, B. (2013). Dynamic voltage and frequency scaling scheduling algorithm for mixed task set. *Proceedings of 8th IEEE International Conference on Industrial and Information Systems (ICIIS 2013), University of Peradeniya, Kandy, SriLanka, December 2013*, pp. 643-648. **(SCOPUS Indexed)**
5. Digalwar, M., Mohan, S., & Raveendran, B. (2013). Energy Aware Real Time Scheduling Algorithm for Mixed Task Set. *Proceedings of IEEE International Conference on Advanced Electronic Systems (ICAES 2013)*, pp. 325-327.
6. Digalwar, M., Raveendran, B. K. & Mohan, S. LAMCS: A Leakage Aware DVFS based Mixed Task Set Scheduler for Multi-Core Processors. *Journal of Sustainable Computing, Elsevier Publication* **(Revisions Communicated)**.

Biography

Biography of the Candidate

Ms. Mayuri Digalwar received M.E degree from Birla Institute of Technology and Science (BITS), Pilani, India in 2009. She is currently pursuing Ph.D. degree in BITS, Pilani and is lecturer in the same institute. She has over 8 years of teaching experience. Her research interest includes real time scheduling algorithms, energy efficient scheduling and energy efficiency issues in modern multi-core processors.

Biography of the Supervisor

Prof. Sudeept Mohan completed his PhD in Electrical Engineering from the Birla Institute of Technology and Sciences (BITS), Pilani. He also holds a Masters (MSc.) degree in Physics and Master of Engineering (M.E) in Electronics and Control from the same institute. He has a teaching and research experience of over twenty years at BITS, Pilani. Currently he is attached to the Computer Science department at BITS as Professor. His research interests include automatic controls and robotics.

Biography of the Co-supervisor

Dr. Biju K Raveendran is Assistant Professor in Department of Computer Science and Information Systems in Birla Institute of Technology and Science, Pilani, K. K. BIRLA Goa campus. He is also heading the Computer Centre Unit which is the central computing infrastructure for the complete campus. He obtained his Ph.D. (Computer Science) from Birla Institute of Technology and Science, Pilani in 2003. His research interests are high performance computing, distributed operating systems, real-time and embedded operating systems, high performance architecture.