

**Design of Energy Efficient Schedulers for  
Multicore Hard Real Time Systems**

**THESIS**

submitted in partial fulfillment  
of the requirements for the degree of

**DOCTOR OF PHILOSOPHY**

by

**GAWALI SHUBHANGI KRUSHNACHANDRA**

under the supervision of

**Dr. BIJU K. RAVEENDRAN**

and co-supervision of

**Prof. BHARAT M. DESHPANDE**



**BITS Pilani**  
Pilani|Dubai|Goa|Hyderabad


**BIRLA INSTITUTE OF TECHNOLOGY AND  
SCIENCE, PILANI**


**March 2018**

**BIRLA INSTITUTE OF TECHNOLOGY AND  
SCIENCE, PILANI**

## **Certificate**


This is to certify that the thesis entitled 'Design of Energy Efficient Schedulers for Multicore Hard Real Time Systems' and submitted by **GAWALI SHUBHANGI KRUSHNACHANDRA**, ID.No. **2011PHXF0020G** for award of Ph.D. of the Institute embodies original work done by her under our supervision.

Signature of the Supervisor :   
Name in capital letters : **Dr. BIJU K. RAVEENDRAN**  
Designation : **ASSISTANT PROFESSOR**  
Date : **27<sup>th</sup> March 2018**

Signature of the Co-supervisor :   
Name in capital letters : **Prof. BHARAT M. DESHPANDE**  
Designation : **ASSOCIATE PROFESSOR**  
Date : **27/03/2018**

# Declaration

I, Gawali Shubhangi Krushnachandra, hereby declare that the work presented here in the thesis titled, 'Design of Energy Efficient Schedulers for Multicore Hard Real Time Systems' is a bonafide research work done by me and it has not been submitted previously in part or in full to this University or any other University or Institution for award of any degree. Any literature work cited within this thesis has given due acknowledgement and is listed in bibliography.

Signature of the student : 

Name of the student : GAWALI SHUBHANGI  
KRUSHNACHANDRA

ID number of the student : 2011PHXF0020G

Date : 27<sup>th</sup> MAR 2018

*Dedicated to my family  
for their continuous support...*

## Acknowledgement

First and foremost I thank Lord Ganesha and Goddess Saraswati for the blessings they bestowed upon me and for giving me the strength and wisdom to achieve this dream.

This doctoral thesis is in its current form due to the assistance and encouragement of several people. It is a pleasure to express my sincere thanks to all those who helped me for the success of this study and made it an unforgettable experience.

I would like to express my deepest gratitude to supervisor, Dr. Biju K. Raveendran. It has been an honour to be his first Ph.D. student. I appreciate all his contributions of time, excellent guidance, ideas, care and patience to make my research journey from start to conclusion an insightful, thought provoking and motivational learning process. The joy and enthusiasm he has for his research was contagious and inspirational for me even during tough times in the Ph.D. pursuit.

I have been extremely lucky to have co-supervisor, Prof. Bharat M. Deshpande, who has provided his valuable guidance and consistent support throughout research work. His apt suggestions in completing my thesis are highly appreciated.

I would like to thank the members of Doctoral Advisory Committee, Dr. Sanjay K. Sahay and Prof. Neena Goveas, for their valuable time, guidance, critical suggestions and comments for overall improvement of research work.

I am grateful to Prof. Souvik Bhattacharyya, Vice-Chancellor, BITS Pilani, Prof. G. Raghurama, Director, BITS-Pilani, K. K. Birla Goa Campus, Prof. Sasikumar Punnekkat, former Director, BITS-Pilani, K. K. Birla Goa Campus, Prof. S. K. Verma, Dean, Academic Research Division, BITS-Pilani, Prof. P. K. Das, Associate Dean, Academic Research Division, BITS-Pilani K. K. Birla Goa Campus and the members of Doctoral Research Committee of Dept. of CS&IS, BITS-Pilani, K. K. Birla Goa Campus for providing administrative support, conducive atmosphere and adequate facilities to carry out my research efficiently.

Completing this work would have been all the more difficult without in-depth discussions, any time help of Dr. Ramprasad Joshi, Rajendra Roul and Geeta Patil.

I owe a lot to all my teachers, colleagues, friends and relatives who have helped me directly or indirectly at different stages of my research work.

I remain indebted to my parents, Seema and Madhav Ghubade and my mother-in-law Vandana for their blessings, family support and motivation. I am thankful to my brothers Nilesh, Nikhil and my sister-in-law Gurpreet for encouraging me with their best wishes.

From the bottom of my heart I thank my daughter Spandana and my husband Krushnachandra for their love and encouragement throughout this work.

**Shubhangi**

# Abstract

Energy efficiency is one of the most important design considerations of modern battery operated real time systems. Static and dynamic energy components are the most prominent parameters affecting energy consumption while scheduling. These energy consumptions are mainly reduced by slowdown and/or shutdown techniques like Voltage and Frequency Scaling (VFS) and procrastination respectively. As hard real-time systems are designed for Worst Case Execution Time (WCET) of jobs, the time difference between WCET and Actual Execution Time (AET) can be used dynamically at run time for shutdown/slowdown to save energy further. Dynamic VFS (DVFS) is a well addressed area of work for uniprocessor systems. Though many researchers used Dynamic Procrastination (DP) to improve shutdown duration, still there exist a scope of improvement without increasing complexity of the scheduler. The real challenge lies in opting DVFS or DP optimally for combined benefits. In Multi-core (MC) systems, the optimal energy performance is achievable by accumulating distributed idle intervals at various cores together and converting it into shutdown/slowdown. This can be achieved by migrating jobs between cores. In MC systems, allocation plays an equally important role as scheduling. Most of the existing allocations are based on optimizing number of cores. In this work, we propose a task allocation strategy - Modified First Fit Bin Packing (**MFFBP**) which considers energy along with number of cores. Experimental results showed that MFFBP increased the shutdown duration of 'Earliest Deadline First with shutdown' which resulted in 13.43% static and 8.42% dynamic energy savings. Producing a valid schedule by meeting all job deadlines with least possible energy consumption is achievable when

the system has minimum idle time. This can be achieved by aggressively converting idle intervals into shutdown whenever possible and active otherwise.

In this thesis we propose four schedulers for different classes of cores namely **DPS** - for the cores supporting shutdown, **DPVFS** - for the cores supporting shutdown with DVFS, **OASIS** - for cores with shutdown and migration and **HANDT** - for cores with shutdown, DVFS and migration. A new framework **SMART** - Simulator for Multicore hArd Real Time systems is designed to schedule and analyze energy parameter. Experimental result shows that on an average, DPS offers 10.7%, 4% and 6.2% reduction in static, dynamic and total energy consumptions respectively in comparison with Conventional DPS (ConvDPS). DPVFS reduces static, dynamic and total energy by {84.54%, -1.45%}, {-0.38%, 32.7%}, {33.2%, 18.8%} over ccEDF and DPS algorithms respectively. OASIS reduces static and overall energy consumptions by {8%, 0.7%} and {3.88%, 1.2%} over ConvDPS and DPS schedulers respectively. HANDT reduces static, dynamic and overall energy by {50%, -3%, 0.83%, -5.33%}, {-5.5%, 6%, 1.46%, 5.3%}, {23%, 2.4%, 1%, 1.3%} over DPS, DPVFS and OASIS algorithms respectively. The work presented in this thesis provides insight of scheduling techniques in MC-HRTS from the perspective of energy saving and analytical view for further enhancements in the area of real time system.



# Contents

<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xiv</b>
<b>List of Abbreviations</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Motivation . . . . .	5
1.3 Problem Statement . . . . .	6
1.4 Research Goals . . . . .	13
1.5 Contributions . . . . .	14
1.6 Thesis Outline . . . . .	15
<b>2 Literature Survey</b>	<b>18</b>
2.1 Introduction . . . . .	18
2.2 Fundamentals . . . . .	20
2.3 Scheduling in Uniprocessor HRTS . . . . .	23
2.3.1 Shutdown Techniques . . . . .	24
2.3.2 Slowdown Techniques . . . . .	25
2.4 Scheduling in MC-HRTS . . . . .	27
2.4.1 Task Allocation Methods . . . . .	29
2.4.2 Task Scheduling Methods . . . . .	31

---

2.5	Summary . . . . .	40
<b>3</b>	<b>System Model</b>	<b>43</b>
3.1	Introduction . . . . .	43
3.2	SMART - A Simulator for MC-HRTS . . . . .	43
3.3	Platform Model . . . . .	44
3.4	Task Model . . . . .	44
3.5	Evaluation Parameters . . . . .	47
3.6	Energy Model . . . . .	51
3.7	Summary . . . . .	55
<b>4</b>	<b>Energy Efficient Scheduling in MC-HRTS</b>	<b>56</b>
4.1	Introduction . . . . .	56
4.2	Task Allocation . . . . .	57
4.2.1	MFFBP Algorithm . . . . .	58
4.2.2	Motivating Example . . . . .	60
4.2.3	Analysis of MFFBP algorithm . . . . .	61
4.2.4	Experimental evaluation . . . . .	63
4.3	Energy Efficient Dynamic Schedulers . . . . .	65
4.3.1	DPS Scheduler . . . . .	66
4.3.1.1	Working of Conventional Static and Dynamic Procrastination Scheduler . . . . .	67
4.3.1.2	DPS algorithm . . . . .	69
4.3.1.3	Motivating Example . . . . .	71
4.3.1.4	Analysis of DPS Algorithm . . . . .	72
4.3.1.5	Experimental Evaluation . . . . .	77
4.3.2	DPVFS Scheduler . . . . .	84
4.3.2.1	Working of ccEDF . . . . .	85

---

4.3.2.2	DPVFS algorithm . . . . .	86
4.3.2.3	Motivating Example . . . . .	89
4.3.2.4	Analysis of DPVFS Algorithm . . . . .	92
4.3.2.5	Experimental Evaluation . . . . .	96
4.4	Summary . . . . .	103
<b>5</b>	<b>Scheduling using Migration</b>	<b>105</b>
5.1	Introduction . . . . .	105
5.2	Energy Efficient Dynamic Schedulers . . . . .	106
5.2.1	OASIS Scheduler . . . . .	107
5.2.1.1	OASIS algorithm . . . . .	108
5.2.1.2	Motivating Example . . . . .	114
5.2.1.3	Analysis of OASIS Algorithm . . . . .	118
5.2.1.4	Experimental Evaluation . . . . .	126
5.2.2	HANDT scheduler . . . . .	138
5.2.2.1	HANDT algorithm . . . . .	141
5.2.2.2	Motivating Example . . . . .	147
5.2.2.3	Analysis of HANDT Algorithm . . . . .	153
5.2.2.4	Experimental Evaluation . . . . .	160
5.3	Summary . . . . .	167
<b>6</b>	<b>Conclusion and Future Directions</b>	<b>170</b>
6.1	Summary of results . . . . .	170
6.2	Future scope . . . . .	173
	<b>Publications</b>	<b>176</b>
	<b>Biographies</b>	<b>177</b>

**Bibliography****179****Index****190**

# List of Figures

2.1	Outline of the Survey . . . . .	19
4.1	Percentage of active, idle and shutdown period for varying utilizations . . . . .	64
4.2	Static energy consumption per unit for different utilizations . .	64
4.3	Total energy consumption per unit for different utilizations . .	65
4.4	(a) Schedule using Static Procrastination with $AET = WCET$ (b) Schedule using Static Procrastination with $AET = 80\%$ $WCET$ (c) Schedule using Conventional DPS . . . . .	68
4.5	(a) Schedule without procrastination (b) Schedule with DPS .	72
4.6	AP and PI . . . . .	73
4.7	Percentage of active, idle and shutdown period for varying utilizations . . . . .	78
4.8	Static energy consumption per unit for different utilizations . .	78
4.9	Total energy consumption per unit for different utilizations . .	79
4.10	Percentage of active, idle and shutdown period for varying utilizations . . . . .	80
4.11	Static energy consumption per unit for different utilizations . .	81
4.12	Dynamic energy consumption per unit for different utilizations	81
4.13	Decision making energy consumption per unit for different utilizations . . . . .	83

---

4.14	Total energy consumption per unit for different utilizations . .	83
4.15	Schedule using ccEDF . . . . .	85
4.16	(a) Schedule with DPVFS (b) Schedule with DPS . . . . .	91
4.17	Static energy consumption per unit time for different utilizations	97
4.18	Dynamic energy consumption per unit time for different utilizations . . . . .	97
4.19	Decision making energy per unit time for different utilizations	98
4.20	Shutdown energy per unit time for different utilizations . . . .	100
4.21	Idle state energy per unit time for different utilizations . . . .	100
4.22	Total energy consumption per unit time for different utilizations	101
4.23	Total energy consumption per unit time for different shutdown thresholds . . . . .	101
5.1	Core State transitions . . . . .	116
5.2	Percentage of active, idle and shutdown period for varying utilizations . . . . .	127
5.3	Static energy consumption per unit time for varying utilization . . . . .	128
5.4	Dynamic energy consumption per unit time for varying utilization	128
5.5	Number of procrastination decision making points for varying utilizations . . . . .	129
5.6	Shutdown overhead per unit time for varying utilizations . .	131
5.7	Total inactive duration energy consumption per unit time for varying utilizations . . . . .	131
5.8	Total energy consumption per unit time for varying utilizations	132
5.9	Percentage of shutdown period over idle period for varying utilizations . . . . .	133

---

5.10	Static energy consumption per unit time for varying utilizations	135
5.11	Dynamic energy consumption per unit time for varying utilizations . . . . .	135
5.12	Number of procrastination decision making points for varying utilizations . . . . .	136
5.13	Shutdown overhead per unit time for varying utilizations . . .	136
5.14	Total inactive duration energy consumption per unit time for varying utilizations . . . . .	137
5.15	Total energy consumption per unit time for varying utilizations	137
5.16	Core State Transitions . . . . .	149
5.17	Shutdown duration for different utilizations . . . . .	161
5.18	Static energy consumption per unit for different utilizations . .	162
5.19	Dynamic energy consumption per unit for different utilizations	162
5.20	Total energy consumption per unit for different utilizations . .	163
5.21	Shutdown duration for different utilizations . . . . .	164
5.22	Static energy consumption per unit for different utilizations . .	165
5.23	Active duration for different utilizations . . . . .	165
5.24	Dynamic energy consumption per unit for different utilizations	166
5.25	Total energy consumption per unit for different utilizations . .	166

# List of Tables

2.1	Ready reckoner for evaluation metrics in task allocations algorithms . . . . .	41
2.2	Ready reckoner for evaluation metrics in basic energy scheduling algorithms for uniprocessor system . . . . .	42
3.1	Algorithms and Techniques used . . . . .	45
3.2	Dynamic energy . . . . .	52
4.1	Task set 1 . . . . .	60
4.2	Task set 2 . . . . .	90
5.1	Task set 3 . . . . .	118
5.2	Task set 4 . . . . .	152
6.1	Ready reckoner for choice of algorithm . . . . .	174



# List of Abbreviations

<b>AET</b>	Actual Execution Time
<b>BP</b>	Bin Packing
<b>BFBP</b>	Best Fit Bin Packing
<b>ccEDF</b>	Cycle Conserving Earliest Deadline First
<b>DP</b>	Dynamic Procrastination
<b>DVFS</b>	Dynamic Voltage/Frequency Scaling
<b>EDF</b>	Earliest Deadline First
<b>FFBP</b>	First Fit Bin Packing
<b>HRTS</b>	Hard Real Time System
<b>MC-HRTS</b>	Multicore Hard Real Time System
<b>MFFBP</b>	Modified First Fit Bin Packing
<b>NFBP</b>	Next Fit Bin Packing
<b>VFS</b>	Voltage/Frequency Scaling
<b>WCET</b>	Worst Case Execution Time
<b>WFBP</b>	Worst Fit Bin Packing

# Chapter 1

## Introduction

### 1.1 Background

Energy efficiency at various levels in real-time systems is one of the most widely addressed research areas as it plays a vital role in prolonging battery life. Some of these systems are working with hard real-time tasks where meeting all task deadlines with minimum energy consumption is a major design consideration [1]. Optimization of energy consumption in uniprocessor, Multi-Core (MC) and Multiprocessor (MP) hard real-time systems can be addressed at various levels of design hierarchies such as technology, circuit, architecture, operating system, compiler and application [1] [2]. Operating system and architecture level energy consumption can be addressed at process execution, scheduling, memory, file systems, synchronization, instruction set architecture, interconnection network, cache memory, voltage and frequency scaling etc. [3]. This thesis addresses energy consumption during process execution and scheduling as it accounts for most of the system level energy consumption.

The major components of energy consumption are dynamic and static. Dynamic energy is due to switching current and static energy is due to

leakage current. Dynamic energy consumption can be reduced by optimizing platform independent parameters like number of preemptions, cache memory impacts etc. Dynamic energy consumption can also be controlled by platform dependent parameters like supply voltage ( $V$ ) and operating frequency ( $f$ ) as dynamic energy consumption of CMOS circuit is proportional to  $V^2fC$  and short-circuit energy consumption where  $C$  is the load capacitance [4]. As dynamic energy consumption has quadratic dependence on supply voltage and frequency, Voltage and Frequency Scaling (VFS) is one of the best mechanisms to reduce it. Most of the real-time jobs finish execution well before its Worst Case Execution Time (WCET). This results in generating additional slack at run time. This slack can be utilized by Dynamic VFS (DVFS) techniques to improve dynamic energy saving [5] [6] [7]. The effectiveness of DVFS scheduler depends on the utilization of idle time and runtime slack. DVFS saves dynamic energy at the cost of increase in job execution time. With increase in execution time, the overhead due to preemptions, cache impacts and other decision making increases. This may result in deadline misses if the scaled execution time of the job overshoots the WCET. The cores consume static and dynamic energy in active and idle state. The dynamic energy consumption of a core in idle state is minimum. Both the dynamic and static energy components are negligibly small in shutdown state. Though the switching current reduces with advancement in CMOS technology, the leakage current exponentially increases. This makes static energy equally an important component as dynamic energy in overall energy consumption. Shutting down the core is one of the prominent mechanisms to save static energy consumption. Saving because of shutdown

is proportional to the length of the shutdown duration as each shutdown has hidden overheads. This can be achieved with the help of procrastination techniques [4] [8] [9] [10] [11]. Procrastination uses prior knowledge of the tasks to postpone their execution without missing any timing constraints. Dynamic Procrastination (DP) can increase the shutdown duration as it uses the run time slack for postponement of job execution. It is known that applying DVFS slims down the chances of future shutdown. In MC systems, the unused idle durations are utilized by migrating the tasks which may effectively produce longer idle duration in some cores for shut down and thus reducing overall energy consumption.

Shutting down the core is effective only if the shutdown duration is beyond a threshold duration. DVFS is effective during high utilization with actual execution time lesser than WCET. DVFS is not very effective when the system utilization is below a threshold as it is practically impossible to reduce the frequency below a threshold. Combining these approaches will offer optimal energy saving while satisfying the quality of schedule. In a hard-real time system, quality of the schedule depends on jobs finishing before their deadline. Early execution of a job may result in a non-optimal schedule in terms of energy consumption. Effectively combining DVFS and DP to achieve optimal overall energy savings in MC systems is still a challenge [4] [10] [12] [13] [14].

Most of the applications are generally made up of multiple threads, each performing its intended function. Complex real-time applications like

avionics, automotive and robotics require high performance processors to guarantee timing constraints which results in high energy consumption due to clock frequency. An alternate solution to achieve the same performance is to simultaneously process the threads on multiple cores. This solution guarantees reduced energy consumption without compromising on performance. Multi-core/Multiprocessor systems introduces new challenges like load balancing, load sharing, optimal shutdown, etc. which can be addressed with the help of an efficient real-time scheduler. Load sharing and load balancing techniques like job migration improves the performance, schedulability and fairness of the executing jobs among the cores[15] [16]. Job migration because of load-balancing incurs additional energy and time overheads due to code/data migrations and local cache invalidations. The concept of job migration is particularly used in distributed operating systems and parallel computing domain for fault tolerance and load balancing. Controlled job migration can also be used effectively in MC systems for optimizing energy consumption. The energy - performance ratio of a MC system can be improved on selected cores by combining with various techniques like DVFS and DP. The dynamic scheduler can minimize the number of cores in use while maximizing the efficiency of the cores.

In real time systems, it is difficult to satisfy timeliness constraint along with energy efficiency. The objective of this thesis is to provide solutions to achieve these goals for processors supporting various modes and techniques. This thesis proposes techniques for optimizing energy consumption in homogeneous Multi-Core Hard Real Time System (MC-HRTS) with static

and semi-partitioning allocation of hard and soft affine multi-core system respectively. It uses migration to enhance the effects of DP and DVFS techniques. This is achieved by migrating jobs to other cores and shutting the cores down for maximum possible duration. This utilizes the unproductive idle time of the active core to execute the jobs from other cores and convert idle time to shut-down duration using procrastination techniques. The run time slack is utilized to execute jobs at reduced voltage and frequency. Migration with DP and DVFS helps in increasing the utilization and shutdown duration which results in saving static and dynamic energy consumptions further.

## 1.2 Motivation

Over the decades, speed of computing has increased exponentially and so has the energy requirement. Such increased energy consumptions causes economic, ecological and technical problems. In critical real time systems, this may cause some fatal effects. For some highly reliable battery operated critical systems like avionics, space vehicles, lunar rover etc. the crucial design objective is not only to meet temporal constraint but also energy consumption. With advancement in processor technology, there is a shift from uniprocessor system to multi-core and multiprocessor systems to meet exponentially increasing performance requirements. This resulted in increasing the static energy consumption because of more leakage current per transistor and dynamic energy consumption because of increased number of transistors per core. Multi-core systems which are more complex than a uniprocessor systems offer better scope for energy optimization as each core can be executed in different

state independently at the same time. When one set of cores are powered down, other set can be with reduced voltage/frequency while remaining cores execute in full voltage/frequency. Migration can be utilized for improving energy saving further though it adds more complexity to the system. In homogeneous MC-HRTS, scheduler plays an important role to produce a valid sequence of job execution without missing deadlines. The scheduler has to optimally consume energy especially in battery operated real time systems. A schedule will be optimal if it offers minimum idle time in a processing system as idle time consumes both static and dynamic energies for unproductive work. Existing work in literature addresses this problem with the help of DVFS and DP. It is much easier to address each core separately with a static task allocation model like Bin Packing. The cores can implement DVFS and DP whenever required to optimize the energy consumption. Meeting all deadline constraints with least possible idle time and minimum overheads to achieve optimal energy saving is an important research problem to address. The idle and execution time can be minimum only if the shutdown duration is maximum. The idle time can also be minimized by maximizing execution time with least energy consumption. This thesis addresses the problem of producing a valid schedule with least idle time and maximum shutdown time to minimize energy consumption.

### **1.3 Problem Statement**

This thesis addresses energy consumption during process execution and scheduling as it consumes a major portion of the system level energy.

In MC-HRTS, the design of task allocation and task scheduling policies significantly affect the energy consumption of the system. The aim of this thesis is to design energy efficient task allocation and scheduling algorithms for MC-HRTS by investigating the state of art algorithms existing in this field of study. This research also aims at improving the energy saving of MC-HRTS without performance degradation by using various static and dynamic energy optimization techniques like DP and DVFS. This thesis aims to minimize the overall energy consumption during real time task execution and scheduling subject to both schedulability and energy constraints. The five objectives of this thesis as optimization problem for homogeneous MC-HRTS with M cores supporting various modes and techniques are expressed as follows:

**Objective 1: Task allocation model**

To optimize the procrastination duration by providing allocation that maximizes the probability of core shutdown in homogeneous MC-HRTS. Formally the problem is defined as:

For some set of cores having higher period tasks,

Maximize

$$SD_j(P_{low}, E_{low}) \quad (1.1)$$

where  $P_{low}$  and  $E_{low}$  are the period and execution time of the lowest period task allocated to core  $C_j$ ,  $SD_j$  is the shutdown duration of core  $C_j$ .

Subject to

$$0 \leq U_j \leq 1 \quad \forall j = 1 \text{ to } M \text{ cores} \quad (1.2)$$

where  $U_j$  is the utilization function of core j.



and for the remaining cores no shutdown is possible.

The tasks should be allocated to the core such that they are schedulable on a uniprocessor system. This optimization aims to maximize the shutdown duration of cores by allocating higher period tasks to some set of cores. In a periodic tasks set, the largest idle duration possible in a schedule is  $2P_{low} - 2E_{low}$ . By allocating higher period tasks together, one improves the chances of maximizing the shutdown duration of cores. This minimizes the overall energy consumption. This is achieved without increasing the number of cores required by First Fit Bin Packing allocation. An approximation algorithm - MFFBP is proposed to achieve the same. MFFBP algorithm is explained in Chapter 4.

### Objective 2: Dynamic Procrastination model

The optimization problem is to minimize overall energy consumption of system supporting various power modes like active, idle and shutdown for a MC-HRTS. Formally the problem is defined as:

Maximize

$$\sum_{j=1}^M SD_j(\sigma) \quad (1.3)$$

Minimize

$$\sum_{j=1}^M E_j(V, f) \quad (1.4)$$

Subject to schedulability condition in each core  $C_j$ :

$$\forall J_k \in C_j, J_k.ct \leq J_k.d \quad (1.5)$$

where  $\sigma$  is the dynamic slack produced due to early completion of jobs and procrastination,  $V$  is the supply voltage for different states of core: 0 for shutdown,  $V_{min}$  for idle and  $V_{max}$  for active,  $f$  is the fixed operating frequency,  $E_j$  is the energy consumption function of core  $j$ ,  $J_k$  is the job allocated to core  $C_j$ ,  $ct$  is the completion time and  $d$  is the absolute deadline of job  $J_k$ . This optimization aims to maximize the shutdown duration and thus minimizes the static energy consumption. A scheduling algorithm - DPS is proposed for achieving the same. The DPS algorithm is explained in Chapter 4.

### **Objective 3: Dynamic Voltage/Frequency Scaling and Procrastination model**

The optimization problem is to minimize overall energy consumption of system supporting various power modes and discrete levels of voltage and frequencies for a MC-HRTS. Formally the problem is defined as:

$$\forall_{j=1}^M C_j, \text{ Maximize} \quad SD_j(\sigma) \quad (1.6)$$

and Minimize

$$ID_j(\sigma) \quad (1.7)$$

where  $\sigma$  is the dynamic slack produced due to early completion and DP of jobs at discrete voltage/frequency levels.

Overall energy consumption =

Active state energy + Idle state energy + Shutdown state energy =

$$\sum_{j=1}^M (U_j / f_x) \cdot (E_j(V_x, f_x)) + E(V_{min}, f) * \sum_{j=1}^M ID_j(\sigma) + \Delta * \text{No. of shutdowns} \quad (1.8)$$

Subject to

$$f_{min} \leq f_x \leq f_{max} \quad (1.9)$$

$$V_{min} \leq V_x \leq V_{max} \quad (1.10)$$

and equation 1.5.

This optimization aims to minimize the overall energy consumption. Due to scaled voltage and frequency, the overall active duration increases. This reduces the overall idle duration. Maximizing shutdown duration also minimizes idle duration. Thus overall idle state energy reduces. Though active duration increases due to scaled frequency, active state energy remains low due to scaled voltage. This inherently causes reduction in overall energy consumption. A scheduling algorithm - DPVFS is proposed for achieving the same. The DPVFS algorithm is explained in Chapter 4.

#### **Objective 4: Dynamic Procrastination and Migration model**

The optimization problem is to minimize overall energy consumption of the system supporting migration and various power modes like active, idle and shutdown for a MC-HRTS. Formally the problem is defined as:

For core  $C_j$  from where the jobs are migrated,

Maximize

$$SD_j(\sigma) \quad (1.11)$$

For core  $C_k$  to which the jobs are migrated,

Minimize

$$ID_k(\sigma) \quad (1.12)$$

where  $\sigma$  is the dynamic slack produced due to early completion, DP and migration of jobs,  $ID_k$  is the idle duration of core k.

Overall energy consumption =

Active state energy + Idle state energy + Shutdown state energy =

$$\sum_{j=1}^M U_j \cdot E_j(V_{max}, f) + E(V_{min}, f) * \sum_{j=1}^M ID_j(\sigma) + \Delta * No. \text{ of shutdowns} \quad (1.13)$$

Subject to equations 1.2 and 1.5.

This optimization aims to minimize the overall energy consumption. Due to fixed frequency, the active state energy remains same even after migration. So the overall energy consumption is affected by idle and shutdown state energy. Even though the shutdown state energy,  $\Delta$  is negligibly small, it consumes some energy for shutting down and waking up the core. Thus maximizing shutdown duration minimizes idle duration and overall energy. A scheduling algorithm - OASIS is proposed for achieving the same. The OASIS algorithm is explained in Chapter 5.

### **Objective 5: DVFS, DP and Migration model**

The optimization problem is to minimize overall energy consumption of system supporting various power modes, discrete levels of voltage/frequencies and migration for a MC-HRTS. Formally the problem is defined as:

For core  $C_j$  from where the jobs are migrated, Maximize

$$SD_j(\sigma) \quad (1.14)$$

For core  $C_k$  to which the jobs are migrated and for cores having insufficient duration for shutdown, Minimize

$$ID_k(\sigma) \quad (1.15)$$

where  $\sigma$  is the dynamic slack produced due to early completion of jobs, DP, migration at discrete voltage/frequency levels.

Overall energy consumption =

Active state energy + Idle state energy + Shutdown state energy =

$$\sum_{j=1}^M (U_j / f_x) \cdot (E_j(V_x, f_x)) + E(V_{min}, f) * \sum_{j=1}^M ID_j(\sigma) + \Delta * \text{No. of shutdowns} \quad (1.16)$$

Subject to

$$f_{min} \leq f_x \leq f_{max} \quad (1.17)$$

$$V_{min} \leq V_x \leq V_{max} \quad (1.18)$$

and equations 1.2 and 1.5.

This optimization aims to minimize the overall energy consumption. Dynamic procrastination with migration increases overall shutdown duration. Whenever it is not possible to shutdown, the core is either kept active by migrating jobs from other cores or it slowdown. This increases the overall active duration. Maximizing overall shutdown and active duration reduces the

overall idle duration. This causes reduction in overall idle state energy. Though active duration increases because of scaled frequency, active state energy consumption remains low due to scaled voltage. Thus overall energy consumption reduces. A scheduling algorithm - HANDT is proposed to achieve the same. HANDT algorithm is explained in Chapter 5.

## 1.4 Research Goals

This research focuses on minimizing the energy consumption during task execution and scheduling in MC-HRTS. Following were the research goals:

***Research Goal 1:*** To design a task allocation algorithm to increase the shutdown duration for MC systems.

***Research Goal 2:*** To design a scheduling algorithm to reduce static energy consumption for hard real time tasks having hard affinity.

***Research Goal 3:*** To design a scheduling algorithm to reduce static and dynamic energy consumption for hard real time tasks having hard affinity.

***Research Goal 4:*** To design a scheduling algorithm to reduce static energy consumption for hard real time tasks having soft affinity.

***Research Goal 5:*** To design a scheduling algorithm to reduce static and dynamic energy consumption for hard real time tasks having soft affinity.

**Research Goal 6:** To design a simulator for real time task scheduler for performance evaluations of developed methods and analyzing various energy related parameters.

## 1.5 Contributions

The major contribution of this thesis is the design of new task allocation mechanism and energy efficient scheduling algorithms for MC-HRTS. The thesis also offers a framework for analysis of these algorithms. The algorithms are as follows:

**Algorithm 1: MFFBP** is designed for achieving Research Goal 1 using Bin Packing approximation technique. This work contributes to the material in Chapter 4.

**Algorithm 2: DPS** is designed for achieving Research Goal 2 using DP technique. MFFBP and DPS algorithms are published in paper titled “DPS: A Dynamic Procrastination Scheduler for Multi-core/Multi-processor Hard Real Time Systems”, International conference on Control, Decision and Information Technologies (CoDIT), pages 286-291, IEEE, 2016. This work contributes to the material in Chapter 4.

**Algorithm 3: DPVFS** is designed for achieving Research Goal 3 using DP and DVFS techniques. This contribution is published in paper titled

“DPVFS: A Dynamic Procrastination cum DVFS Scheduler for Multicore Hard Real Time Systems”, International Journal of Embedded system (IJES), Inderscience publication, 2017 (in press). This work contributes to the material in Chapter 4.

**Algorithm 4: OASIS** is designed for achieving Research Goal 4 using DP and migration techniques. This work is under review and contributes to the material in Chapter 5.

**Algorithm 5: HANDT** is designed for achieving Research Goal 5 using DP, DVFS and migration techniques. This work is under review and contributes to the material in Chapter 5.

**Simulator:** To achieve Research Goal 6, a Simulator for MC hArd Real Time system (**SMART**) is developed to analyze the energy efficiency of the proposed algorithms along with seminal algorithms from literature and a test bed is developed to verify and validate various algorithms. This work is under review and contributes to the material in Chapter 3.

## 1.6 Thesis Outline

Rest of the dissertation is structured as follows:

**Chapter 2 - Literature Survey** - This chapter explains the state of the art work in energy aware system and energy consumption issues in it. It



presents an extensive work related to DP, DVFS and combination of these techniques along with migration wherever applicable in uniprocessor and MC-HRTS.

**Chapter 3 - System Model** - We present our simulator ‘SMART’ in this chapter. We also explain the platform, task and energy models designed for analyzing various scheduling algorithms.

**Chapter 4 - Energy Efficient Scheduling in MC-HRTS** - This chapter illustrates the energy efficient approaches in a MC-HRTS. It discusses the design of proposed allocation algorithm MFFBP and two novel schedulers DPS and DPVFS. The developed algorithms are analyzed for schedulability, correctness and complexity. Experimental validation and comparison of existing and developed algorithms were carried out on SMART. This chapter also presents details of our experimental results.

**Chapter 5 - Energy Efficient Scheduling in MC-HRTS with Migration** - This chapter illustrates the energy efficient approaches in a MC-HRTS with Migration. This chapter describes the design of two novel schedulers OASIS and HANDT. The developed algorithms are analyzed for schedulability, correctness and complexity. Experimental validation and comparison of existing and developed algorithms were carried out on SMART. This chapter also presents experimental results.

**Chapter 6 - Conclusion and Future Directions** - This chapter consolidates conclusions of our work, its limitations and future research scope.

# Chapter 2

## Literature Survey

### 2.1 Introduction

In order to understand various techniques used for energy optimization in real time systems while maintaining the timeliness constraint, extensive survey has been conducted. In this chapter the features of real time systems, parameters associated with energy consumption and various issues in it are explained. This chapter also presents a survey on state-of-the-art techniques for energy minimization in real time task scheduling. Figure 2.1 shows outline of the survey where boxes represent primary areas of study. The survey begins with identifying the significant factors affecting energy consumption, considered in current work. Followed by this, various energy efficient scheduling techniques in uniprocessor hard real time systems like clock gating, slowdown and shutdown are analyzed in detail. Gating and slowdown are considered as dynamic energy saving techniques and shutdown is considered as static energy saving technique while scheduling. The survey moves on to MC system. For MC system, conventional bin packing approximation methods are studied for task allocation. For task scheduling, slowdown and shutdown techniques are studied for energy optimization along with conventional scheduling techniques. These techniques are analyzed for systems that support migration and for

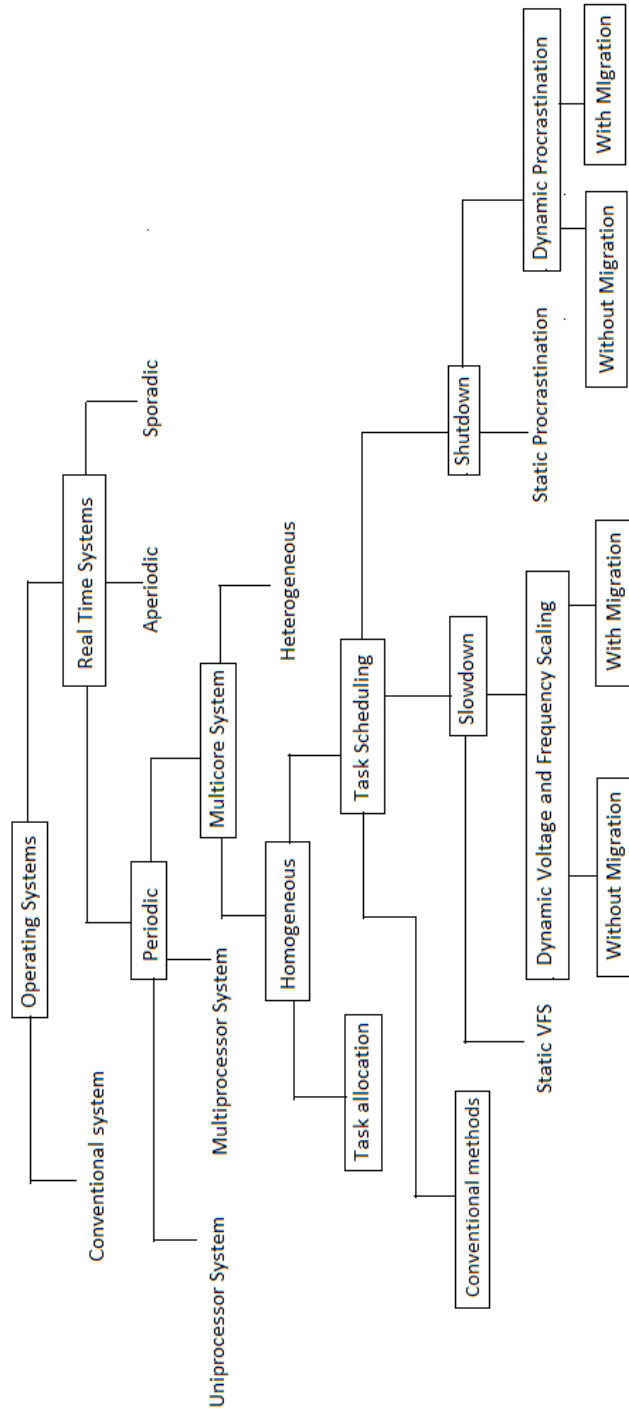


Figure 2.1: Outline of the Survey

the systems which do not support migration. The effect of slowdown and shutdown techniques are analyzed in detail when combined to save both static and dynamic energy. The survey ends with summarizing the combined effect of these techniques adopted in the present work.

## **2.2 Fundamentals of Energy Efficient Real Time Systems**

Real-time systems are characterized by computational activities with strict timing constraints to meet in order to achieve the desired behavior. A typical timing constraint on a real time task is the deadline, which represents the time before which it should complete its execution without causing any damage to the system. Depending on the consequences of a missed deadline, real-time tasks are usually distinguished in three categories: hard, firm and soft [17]. A real-time task is said to be hard if missing its deadline may cause catastrophic consequences on the system under control. For e.g. flight control system, air traffic control system, pace maker, military applications etc. A real-time task is said to be firm in which the computation becomes obsolete on missing the deadline. For e.g. weather forecast system. A real-time task is said to be soft if missing its deadline has still some utility for the system, although causing a performance degradation. For e.g. electronic games, multimedia systems, audio-video streaming. Another timing constraint on a real-time task is based on the regularity of its activation. In particular, tasks can be classified as periodic, aperiodic and sporadic. Periodic tasks consist of an infinite sequence of identical activities, called instances or jobs, that are regularly

activated at a constant rate. The activation of an aperiodic task is an event to which the task respond. These tasks also have timing constraints with soft deadlines or sometimes no deadline. Sporadic tasks are also event based with jobs separated with minimum inter-arrival time, thus with hard deadlines. Tasks are also categorized based on precedence and resource constraints. Tasks whose progress is not dependent upon the progress of other tasks are termed as independent task. Interdependent tasks interact by communication and precedence relationships. A task is blocked if the requested resource is unavailable. This work focuses on real time systems with independent periodic tasks with hard deadlines without any resource constraints. Each task  $T_i$  of a task set  $T$  is represented by  $\{\phi_i, P_i, C_i, D_i\}$  where  $\phi_i$  is phase,  $P_i$  is period,  $C_i$  is WCET and  $D_i$  is relative deadline of the task [18]. All tasks are assumed to be preemptive, in-phase with  $P_i = D_i$ . The  $k^{th}$  job of task  $T_i$  is represented by  $J_{i,k} : \{a_{ik}, c_{ik}, d_{ik}\}$  where  $d_{ik} = a_{ik} + D_i$ . The utilization of task  $T_i$  is calculated as  $U_i = C_i/P_i$  which is the fraction of core time used by  $T_i$ .

Design of hard real time systems need to take care of all job deadlines with minimal energy consumption. Energy consumption while running a system is because of decision making, preemptions, cache filling/invalidating and job execution activities. These activities consume static and dynamic energies. Static energy is due to leakage current and dynamic energy is due to switching of gate from one state to another. The energy consumption due to charging and discharging of gates on a CMOS core ( $E_{switch}$ ) and the short-circuit power consumption ( $E_{sc}$ ) contribute to dynamic energy consumption ( $E_{dynamic}$ ) [19].  $E_{switch}$  is computed as  $V^2 f C$  where  $V$  is the supply voltage,  $f$  is the maximum

frequency for job execution and  $C$  is the load capacitance.  $E_{sc}$  is proportional to the supply voltage [5] [6] [7]. The shorter channels results in exponential increase in leakage current of the CMOS transistor which increases static energy consumption [4] [9] [20]. The leakage current is 0.01A/m for the 130nm and is 3A/m for 45nm technology [4]. In [21], authors gives the ratio of static to dynamic energy for various technologies. They analyzed that when activity is very low (0.01) ratio of static energy to dynamic energy is 0.3 for  $V=0.6$  volts, and 0.5 for  $V=1.2$  volts. They also observed that even technologies with reduced leakage currents (PTM 20nm, 16nm, 14nm, and 65nm) have static energy consumption comparable with dynamic energy for low activity of a gate. Thus static energy is equally an important component as dynamic energy in overall energy consumption. The major contributors of static energy are the sub-threshold leakage current ( $I_{sub}$ ) and the reverse bias junction current ( $I_{rev}$ ) which increases significantly with adaptive body bias voltage ( $V_{bs}$ ) [4]. The static energy consumption ( $E_{static}$ ) due to  $I_{sub}$  and  $I_{rev}$  is computed as  $E_{static} = V.I_{sub} + V_{bs}.I_{rev}$ . In idle state, the cores consume static energy and nominal dynamic energy. Both the energy components are negligibly small in shutdown state. As static energy in idle state is to keep the transistors on, it do not contribute in performing any significant work. Thus it is preferred to keep the core in shutdown state instead of idle to reduce static energy consumption. This saving is proportional to the length of shutdown duration as each shutdown has hidden overheads, like core shutdown, wakeup energy etc. [4] [8] [9] [10] [11].

## 2.3 Energy Efficient Scheduling Techniques in Uniprocessor HRTS

Seminal real time scheduling techniques in use are Rate Monotonic (RM), Earliest Deadline First (EDF) and Least Laxity First (LLF). RM is task level fixed priority scheduling algorithm in which the job with smallest period is given the highest priority for execution. The schedulability condition of RM is, the total utilization is not more than  $n(2^{1/n} - 1)$  where  $n$  is the number of tasks. EDF is task level dynamic and job level fixed priority scheduling algorithm in which the job with earliest deadline is given the highest priority for execution. LLF is a job level dynamic priority scheduling algorithm in which the job with least slack time is given highest priority for execution. Both EDF and LLF offer 100% schedulability. In LLF, run time slack computation overhead affects the performance of the system. For hard real time systems, these algorithms aim at completing the execution of all jobs before their absolute deadlines. Amongst all, deadline driven scheduling policy like EDF guarantees schedulability of all jobs with comparatively less overhead. Thus various existing energy aware deadline driven scheduling variants are explored.

Clock gating and power gating are the most common architecture level energy optimization techniques available [22]. In gating techniques, clock and power supply of the system are gated off resulting in reduced energy consumption. In clock gating, the clock of a unit in the system which is left unused is turned off. This prevents the transistors from switching, leading to zero dynamic energy consumption. Power gating is one of the techniques to conquer static and



dynamic energy consumptions. In power gating, the supply voltage of unused units is cut off to prevent the power dissipation due to sub-threshold leakage current. In MC systems, gating techniques is exploited to gate the power or clock, of one or multiple cores. These techniques imposes the overhead of re-enabling the unit which may affect performance of the system.

### 2.3.1 Shutdown Techniques

The static energy consumed during the idle state of a core is for keeping the transistors on. This do not contribute in performing any significant work. In [20], it is reported that the energy consumption when the processor is in the idle state can be in the order of  $10^3$  compared to when the core is in shutdown state. To reduce static energy, various shutdown techniques are proposed when cores are not in use [4] [23] [24]. The most widely used mechanism is ‘Procrastination’ in which the idle bursts are combined to form sufficient idle duration to shutdown [4] [8] [9] [11] [20] [23] [25] [26] [27] [28] [29]. Procrastination is one of the most efficient techniques to optimize the energy consumption during idle period by delaying the task execution and increasing the shutdown duration while meeting all timing requirements. This incurs additional time and energy because of shutdown and wakeup overheads. Procrastination technique is applicable to the system that supports various core states (active, idle, shutdown, sleep, hibernate etc). Various off-line and on-line procrastination solutions have been proposed for energy savings in real-time under different application/device settings [23]. Earlier work on procrastination is based on precomputed (static) procrastination intervals by considering WCET [4] [23]. In this method, the latest start time for

every job is precomputed with all timing constraints [4]. Most of the jobs completes before WCET thereby leaves some slack before its deadline. With this method, the precomputed idle period for a core is not the maximum procrastinated interval and thus underestimates the procrastinated shutdown duration. On-line (dynamic) procrastination algorithms use this slack to increase shutdown duration of the processor further [9] [10] [11]. In [9], Lee et al. explained how idle duration can be extended by delaying the active period by an interval given by the sum of pseudo execution time of all tasks that are preempted plus the delay of the currently executing earliest deadline task before the start of active period. In [10], Niu and Quan extended the idle interval length by computing the latest start time of the job set without missing the deadline of any future job. Latest start time for every higher priority jobs is computed and the earliest of all these is considered to be the start of active job set.

### 2.3.2 Slowdown Techniques

Due to quadratic dependency of dynamic energy on supply voltage ( $V$ ) and operating frequency ( $f$ ), researchers recommend energy efficient technique called slowdown technique to reduce overall dynamic energy consumption while scheduling [5] [6] [7]. In this technique,  $V$  and  $f$  are scaled down to minimum required voltage and frequency for job execution without violating the timing constraints. Thus this technique is also called Voltage and Frequency Scaling (VFS). The key idea behind VFS is to execute the jobs for longer duration and reduce the idle time of a schedule. The idle time is getting generated because of the low utilization of task set and because of the

early completion of jobs than their predicted WCET. The reduction in idle time results in energy savings. The slowdown methods are applicable to the system that supports discrete levels of voltage and frequency ranges [5] [6] [7]. Enhanced Intel SpeedStep Technology supports processor speeds of 600 MHz to 1.6 GHz with a step of 200 MHz [30]. AMD PowerNow! Technology supports the complete frequency operating range of the processor in use allowing steps of 33 or 50 MHz from an absolute low of 133 or 200 MHz [30].

In [31], Bambagini et al. described a speed modulation technique to achieve the required speed using two discrete values. The method selects the pair of frequencies that minimizes energy consumption. In [5], Pillai and Shin proposed architecture dependent Static (SVFS) and Dynamic Voltage/Frequency Scaling (DVFS) techniques to save dynamic energy consumption during the active period. The SVFS selects the lowest possible  $V$  and  $f$  at WCET that allows EDF/RM schedulers to meet all the deadlines for a given periodic task set [5]. While executing, the jobs may finish well before its WCET which generates slack. This slack can be utilized by executing the future jobs at reduced voltage and frequency. DVFS uses this slack to reduce  $V$  and  $f$  further. In [5], Pillai and Shin proposed two DVFS variants of EDF/RM - Cycle Conserving (CC) and Look Ahead (LA). In CC,  $V$  and  $f$  are recomputed using the Actual Execution Time (AET) of the finished jobs and the WCET of jobs which are there in the queue. They showed 20% to 40% energy saving compared to conventional EDF/RM. The LA technique determines the future computation needs and defers the task execution by setting the operating frequency as low as possible ensuring future deadlines.

## 2.4 Energy Efficient Scheduling Techniques in MC-HRTS

Many prospective areas like wireless network applications, cognitive systems, image recognition units, biomedical systems, automobiles, military applications, various industrial areas such as transportation, automation etc. are in need of Multi-core/Multiprocessor (MC/MP) high performance embedded computing systems. Complex real-time applications like avionics, automotive and robotics are generally made up of multiple threads, each performing its intended function and thus require MC system for high performance with reduced energy consumption. This introduces new challenges like load balancing, load sharing, optimal shutdown etc. which can be addressed with the help of an efficient real-time scheduler. The task sets with total utilization more than 100% needs to be scheduled on MC system. Scheduling problem in MC system is solved in two phases - task allocation and job scheduling [32]. In task allocation, for each task, a core is identified in which the task has to be executed. Several schemes for task allocation on MC system have been proposed. These schemes are broadly classified as global, partitioned and semi-partitioned scheduling schemes [32] [33] [34]. In global scheme, a global ready queue is used and the jobs are allowed to migrate between cores [35]. In this scheme, affinity of a job to a core is very weak which results in increased cache invalidations. The additional overhead because of cache invalidation may result in deadline misses and/or exceeding energy budget of the embedded system. In partitioned scheme, the non-migratable tasks are statically partitioned and assigned to a specific

core for execution [35]. Each core maintains a local ready queue. Due to non-migratable nature, the job remains with the same core on preemption. Due to this, some of the cores may remain idle even when jobs are ready for execution. The worst case timing analysis of partitioned approach is much tighter than the global approach as migration related overheads like cache thrashing are eliminated. Semi-partitioned scheme is a hybrid approach in which, initially the tasks are assigned to cores [36] [37] [38] [39]. During scheduling, the jobs of tasks are allowed to migrate from one core to other. This improves the core utilization and balances the workload among cores. The tightness of worst case timing analysis of semi-partitioned approach lies between global and partitioned approach. The migration in semi-partitioned scheme requires coordination between the cores yielding to high decision making and cache invalidation cost. In real time systems due to timeliness constraint, the overhead of communication between cores need to be avoided. Thus we prefer to use partitioned scheme for jobs having hard affinity and semi-partitioned scheme for the jobs having soft affinity.

In scheduling, the jobs of the allocated tasks are scheduled according to the core's scheduling policy. These policies can be well known uniprocessor scheduling algorithms like RM, EDF, LLF etc. The energy aware scheduling algorithms using slowdown and shutdown techniques can also be applied for the systems that support discrete voltage levels and frequency ranges and multiple modes of core. These techniques are further supported to increase energy saving with migration technique.

### 2.4.1 Partitioned and Semi-Partitioned Task Allocation Methods

The main objective of task allocation algorithms in MC systems is to find the optimal number of cores required for the given task set and assign tasks to the cores. It is considered as packing problem and can be solved as combinatorial optimization. Packing methods like knapsack, utilization balancing algorithm, buddy algorithm, myopic, bin packing, linear programming etc. are some of the popular task assignment algorithms [40]. The allocation of tasks to the core can be achieved using on-line, semi-on-line or off-line algorithms [41]. In on-line algorithms, the task is assigned to the core as soon as it arrives to the system by considering all the tasks which already arrived [41]. This is achieved by carrying out a schedulability test for the newly arrived task with all the allocated tasks. If the newly arrived task passes the schedulability test, it is admitted to the core. The semi-on-line algorithms follow on-line algorithms by keeping an upper bound on rearrangements of allocated tasks [41]. The schedulability test conducted in on-line and semi-on-line algorithms causes run time overhead while scheduling. The off-line algorithms are static algorithms where the number of tasks in the task set and the order in which they are assigned to the core is fixed. Classic Bin Packing approximation is one of the most efficient off-line task allocation used in MC systems where the set of cores are regarded as bins [9]. In this technique, a set of tasks, a set of cores and the utilization of each core are given as input parameters. It divides the task set into sub task sets based on core utilization and allocates it to the cores for scheduling. The optimal

Bin Packing is one of the classic NP-complete problems. The widely used polynomial-time Bin Packing approximation algorithms are First-Fit (FF), Best-Fit (BF), Best-k-Fit (BkF), Next-Fit (NF) and Worst-Fit (WF) [32] [34] [41]. FF allocates a new task to a non-empty core with the lowest index, such that the utilization of the new task along with the utilization of the tasks already allocated to that core, do not exceed the capacity of that core [32] [41]. BF allocates a new task to the non-empty core with smallest capacity available, in which this task can be allocated with a tie breaking strategy as index number [24] [42]. WF is similar to BF, with the difference that it allocates tasks to the cores with the largest capacity available, in which task can be feasibly allocated [24] [42]. FF, BF and WF algorithms allocate the new task to a new core only if it is not fitting in any of the cores whose allocation already started. BkF allocates like BF but considers only the last k open cores for allocation [43]. It allocates the new task to a new core only if it is not fitting in any of the last k cores. In NF, if the new task can fit in the last allocated core, it is allocated to the core. Otherwise a new core is chosen and the task is allocated to that core. Some of the on-line versions of Bin Packing algorithms are First Fit Decreasing (FFD), Refined FFD (RFFD), Modified FFD (MFFD), Best Fit Decreasing (BFD) and Group-X Fit grouped (GXFG) [41]. The performance metric of these algorithms is the ratio of number of cores in approximation algorithms over optimal method. Among all the methods mentioned above, FF has best performance ratio of  $11/9$  [41]. The task assignment can be improved to optimize not only the number of cores but also the energy consumption. Algorithm proposed by Tarek and Hakan allocates tasks to the core and computes CPU speed assignments for

minimizing the total energy consumption [44].

## 2.4.2 Task Scheduling Methods

Most of the MC energy efficient schedulers use DVFS and DP techniques. These techniques are further explored to apply migration technique in MC systems.

### **Dynamic Voltage and Frequency Scaling (DVFS) technique:**

Hakan et al. explained two DVFS schedulers - Dynamic Reclaiming and Aggressive Speed Reduction [6]. In [7], Yang et al. proposed an approximation algorithm for DVFS scheduling on multiprocessor system. In [45], Aydin et al., proposed DRA and AGR algorithms. DRA keeps track of dispatch times of tasks. During runtime, if a task is dispatched earlier, the processor is slowed down to prolong the execution time till the original completion time. AGR estimates the task completion time from the past history and computes the lowest processor speed. In [19], Nassiffe et al. proposed similar solution but assumed that CPU frequency can be selected continuously within a given range. In [46], Chen et al. considered two types of processor models - one with a finite number of discrete processor speeds, and other with an infinite number of continuous processor speeds. They proposed a 2-approximation algorithm and a fully polynomial time approximation scheme. In [47], Ishihara and Yasuura considered DVFS on uniprocessor system with very few discretely variable voltages. They showed the problem of optimizing voltage and frequency in polynomial time with at most two voltages. In [48], Yao et al. considered the energy optimization for



independent jobs with arrival times, deadlines, and WCET on a uniprocessor with variable speeds under the assumption that energy is a convex function of the processor speed. They also considered the case of discretely available processor speeds. They gave an  $\mathcal{O}(N \log^2 N)$  time optimal off-line algorithm where  $N$  is the number of jobs. They also proposed some heuristics for the on-line version of the problem. In [49], Chen et al. applied DVFS method for the jobs with precedence constraints. They considered the case in which speed change is not allowed in the middle of processing a job. They gave fully polynomial-time approximation schemes for two special cases of the problem and also proved the problem to be NP-Complete. For high end compute intensive applications to optimize, make-span Genetic Algorithm (GA) and Immune Genetic Algorithm (IGA) are used in computational grid [50] [51] [52]. These works applies IGA and BAT algorithm to schedule the submitted jobs on the grid nodes for the optimal make span.

In [7], Yang et al. considered the problem of minimizing energy consumption for a chip-multiprocessor with DVFS that can use continuously varying processor speeds with no upper bound. The energy consumed by the processor was assumed to be proportional to the cube of the processor speed. They proposed 2.371-approximation algorithm for DVFS scheduling in MP system. In [53], Zhang et al. considered scheduling on MP system with dependent tasks on fixed number of processors with variable voltage. They described task scheduling phase as voltage selection and formulated it as integer-programming problem that can be solved in polynomial time for the system with continuous voltages. In [54], Shieh and Pong also

provided IP formulation by taking into account the overheads in transition between different voltage levels. They also proposed an on-line heuristic algorithm and compared it with the optimal off-line algorithm. In [55], Chen et al. considered DVFS for shared resources in MC processors. They proposed a method for DVFS of networks-on-chip and last level caches in MC processor designs, where the shared resources form a single voltage/frequency domain. In [56], Min-Allah et al. proposed energy efficient rate monotonic scheduling for MC systems. Their method is to first find the lowest core speed to satisfy deadline to minimize energy. Then it shifts the lightest tasks to different cores to maximize the core utilization. In [57], Zhu et al. considered DVFS for heterogeneous cluster. They proposed an Adaptive Energy-Efficient Scheduling (AEES) for aperiodic and independent real-time tasks on heterogeneous clusters with DVS. The AEES algorithm adjusts the voltages according to the workload conditions of a cluster. When the cluster is heavily loaded, the AEES algorithm considers voltage levels of the new tasks as well as the tasks running currently to meet the deadlines. When the cluster is lightly loaded, the AEES algorithm reduces the voltage levels to conserve energy while satisfying all the deadlines. In [58], Bergamaschi et al. considered an integrated power management unit in a MC processor, which monitors the performance and energy of each core dynamically and change the individual voltages and frequencies to maximize the system performance under a given energy budget.

In [59], Kim et al., noticed that DVFS technique increases the number of preemptions, leading to a higher system utilization and, therefore, higher

energy consumption. To resolve this, they proposed two preemption control DVFS techniques. In [60] and [61], authors reported that DVFS techniques which frequently slows down the execution may result into deadline misses due to voltage switching overheads. All slowdown methods discussed here reduces dynamic energy at the cost of increased execution time which inherently reduces static energy saving. Static energy may also increase if the supply voltage is scaled down beyond threshold. In [62], authors studied the impact of DVFS on core and uncore elements. The uncore is Intels term for the CPU components that are outside but closely associated with the cores (e.g., the last-level cache, memory controller, and interconnect). The authors showed that contrary to conventional wisdom, it is not always energy efficient to run applications at the lowest frequency. This happens because the uncore energy accounts for 74% of the total, out of which uncore static energy constitutes 61% of the total. The frequency at which an application spends the lowest energy depends on how memory bound it is and how many concurrent threads it uses [62]. As per [4], the critical speed for 70nm technology is 0.4 which means that the static energy consumption increases if the supply voltage is scaled beyond 0.4. This enforces limitation on reducing the supply voltage which does not allow minimizing the dynamic energy consumption further.

**Dynamic Procrastination (DP) technique:** In [63], Yang et al. considered the problem of energy efficient real time task scheduling with temperature dependent leakage on a processor [64] [65]. They propose a pattern based approach in which they divide the given time horizon into

several time segments with the same length. In each time segment the processor is in the active mode at the beginning for a fixed amount of time, and it is in the dormant mode at the end for the remaining amount of time. In the active mode the computation is advanced, while the dormant mode is used to reduce the temperature by cooling as well as by reducing leakage energy consumption. Shutdown is feasible only if the procrastinated duration is beyond threshold otherwise the associated overhead of processor shutdown and wakeup can overkill the saving made by shutting down the processor. In [66], Meisner et al., explained how PowerNap concept outperform DVFS approach in realistic applications. In [25], Chen and Kuo proposed a method to simulate the execution of periodic tasks to compute the idle time available until the next deadline. This idle time is used for postponing the tasks to shut the system down. They also proposed virtual blocking which is the maximum blocking that tasks can suffer to extend the procrastination interval. In [67], Awan and Petters proposed accumulation of execution slack and switch the processor off during such intervals under EDF. In [68], Huang et al., proposed an offline analysis that combines DPM and Real-Time Calculus. This method estimates job arrivals, computes CPU idle intervals and modulates between active and sleep states at runtime. In both these methods, the tasks are always executed at the maximum speed. In [69], Energy Saving-Rate Harmonized Scheduler (ES-RHS) is proposed for uniprocessor and MC systems. ES-RHS is based on the notion of harmonization, that aggregates all the processor idle durations together. This allows the processor to be put into shutdown for all idle durations, thus enabling optimal energy savings. In [70], Dsouza et al. proposed ES-RHS+ to enhance the schedulability and

feasibility conditions due to the reduction in the blocking faced by tasks. The basic Rate-Monotonic scheduler is extended in [70] to use a periodic energy saver task, that executes at the highest priority with its execution time. In [71], Fu et al. focuses on reducing the energy consumption of the shared main memory in MC processors by putting it into sleep state when all the cores are idle.

**Combination of slowdown and shutdown techniques:** Slowdown or shutdown alone does not offer optimal energy saving. To balance both static and dynamic energy consumptions, sometimes the tasks are required to be executed at reduced speed and sometimes executing at maximum speed results in increasing idle intervals [10]. In [10], Niu and Quan proposed DVSLK which finds the latest start time of jobs and merges the scattered idle intervals into larger ones such that no jobs misses their deadlines. This algorithm uses static schedule with reduced voltage and frequency for each job defined statically and procrastinates only when no job is ready for execution. In [12], Pagani and Chen adopted a simple and linear-time strategy called Single Frequency Approximation (SFA) on MC system. SFA executes all jobs at critical frequency along with the procrastination scheme. In [4], Jejurikar et al., proposed off-line method Critical Speed DVS with Procrastination (CS-DVS-P) based on the critical speed and task procrastination. CS-DVS-P sorts the tasks by non-decreasing order of relative deadlines and computes the maximum amount of time ( $Z_i$ ) each job can spend in sleep state within its period without leading to any deadline miss. At runtime, when there is no pending job, the core is put in the deepest low-power state until next job

arrival. When a job arrives and the core is still in sleep mode, the core is kept in sleep state for minimum of remaining estimated sleep time and  $Z_i$  of the new job. Both SFA and CS-DVS-P algorithms give optimal result only when all tasks execute for their worst case. In [9], Lee et al. proposed LC-EDF (Leakage-Control EDF), an on-line scheduler. At each job arrival, LC-EDF computes the maximum delay a job can suffer without missing its deadline. When the core becomes idle, LC-EDF computes the maximum extension ( $k$ ) that the first arriving task ( $T_k$ ) can exploit to fully utilize the processor. Then, the sleep time is extended for  $k$  units. If another task  $T_j$  with absolute deadline earlier than  $T_k$  arrives when the core is in sleep state, the procedure is repeated. In [72], Irani et al. extended VFS to include the case in which a processor can go into a sleep state. In sleep state, the processor speed and its energy consumption are 0, but a constant amount of energy is required to bring back the processor into a non-sleep mode. They proposed a 3-approximation algorithm for SVFS. They also proposed an on-line algorithm for VFS.

During low processor utilization, finishing all ready to run jobs at maximum frequency will increase the shutdown duration. During high processor utilization, executing job with reduced voltage and frequency results in dynamic energy saving but it slims down the chances of future shutdown. Combining these approaches will offer optimal energy saving while satisfying the quality of schedule. To effectively combine DVFS and DP to achieve the optimal overall energy savings in MC system is still a challenge [4] [10] [12] [13] [14]. This research worked focuses on combining DVFS and

DP techniques to save both dynamic and static energies while maintaining the schedulability in MC systems.

### **Scheduling soft affine tasks:**

The ability of OS to bind a task to a core or range of more cores is called as affinity. There are two types of CPU affinity - soft and hard. Jobs with soft affinity are allowed to migrate from one core to another whereas jobs with hard affinity are not. Migration of jobs at runtime helps improving utilization if the jobs are cache cold. The migration points depends on compatibility among cores. Binary level compatibility allows jobs to migrate at any point as all the cores will be homogeneous with respect to ISAs. Heterogeneous cores follow source level compatibility with predefined migration points. Unlike distributed systems, cores of MC systems share a common memory and these systems follow internal migration [33] [73]. The data blocks of the migrated job is invalidated in the local cache of the previous core when it gets modified in the new core. The code blocks will never be modified as they are read only. The migration policy of the system tries to address the basic questions like (a) when to migrate the jobs? (b) which jobs to be considered for migration? and (c) where to migrate these jobs? The migration policy should take care of the larger objectives of the system like improving performance utilization, improving response time, fault-tolerance, thermal balancing etc. Most widely used migration strategies are 'Push migration' and 'Pull migration' [33]. Push migration is initiated by the core when it has more workload compared to other cores. [74] The

job satisfying the migration policy is selected for transferring to other core. Pull migration is initiated by the core when it is heavily under utilized. It finds the core from where job(s) can be migrated with least migration overhead. Depending on the code transfer across the cores especially cached data, migration can be whole cache migration or regional cache migration [33]. The overhead of the whole cache migration is very high as it copies the complete code where as the regional migration copies only a part of the code. The remaining part of the code is loaded on demand by the target core.

As energy consumption is becoming an important factor of an embedded system, it is worth an attempt to try migration for energy consumption. The distributed idle/shutdown intervals within a core can be combined together with the help of procrastination techniques. In MC system, the idle/shutdown intervals spread across multiple cores cannot be combined with procrastination. The only way to increase the procrastination duration is to allow migration of jobs. In [75], Chen et al. proposed an optimal polynomial-time scheduling algorithm for the minimization of energy consumption with job migration. They also proposed approximation algorithms for processors with/without constraints on the maximum processor speed. They showed that when there is an upper bound on processor speeds, an artificial-bound approach can be taken to minimize the energy consumption. Migrating current/upcoming jobs aim at reducing overall idle intervals and increasing shutdown duration. The migration helps in reducing idle duration either by executing the migrated job(s) or by converting it into shutdown by migrating some jobs from that core.



Though these techniques work well independently, combining these to get optimized overall energy consumption in MC systems is a challenge. This research worked on optimizing energy consumption using migration with DP and DVFS techniques in homogeneous MC systems with semi-partitioning allocation of tasks having soft affinity.

## 2.5 Summary

A precise survey was conducted to understand the parameters for improving energy savings in real time systems is compiled in this chapter. This chapter explained the features of real time systems, parameters associated with energy consumption, significant factors affecting energy consumption and various issues associated with it. It also presented a survey on state-of-the-art techniques for energy minimization in real time allocation and scheduling. Amongst all, the widely used techniques - DP and DVFS as shutdown and slowdown respectively are studied in detail for uniprocessor and MC systems. For MC system, conventional bin packing approximation methods are studied for task allocation. This work also studied migration technique combined with DP and DVFS techniques for task scheduling. Table 2.1 summarizes the complexity analysis of bin packing approximation algorithms. Table 2.2 summarizes the key papers that has described the basic energy saving scheduling algorithms for uniprocessor system.

Table 2.1: Ready reckoner for evaluation metrics in task allocations algorithms

<b>Bin Packing Approximation Algorithm</b> [32] [34] [41]	<b>Performance ratio</b>	Complexity (Sorting)
First Fit	$11/9 = 1.2$	$\mathcal{O}(N \log N)$
Best Fit	4/3 times FF = 1.6	$\mathcal{O}(N \log N)$
Worst Fit	2	$\mathcal{O}(N \log N)$
Next Fit	2	$\mathcal{O}(N \log N)$

Table 2.2: Ready reckoner for evaluation metrics in basic energy scheduling algorithms for uniprocessor system

Algorithm	Evaluation Metrics	
	Energy parameter	Schedulability
Clock Gating and Power Gating [22]	Static and Dynamic	NA
Static Procrastination [4] [23]	Static and Dynamic	schedulability of base algorithm (EDF/RM)
Dynamic Procrastination [9] [10] [11]	Static and Dynamic	schedulability of base algorithm (EDF/RM)
Static VFS [5]	Dynamic	schedulability of base algorithm (EDF/RM)
ccEDF, ccRM, LAEDF, LARM [5]	Dynamic	schedulability of base algorithm (EDF/RM)

# Chapter 3

## System Model

### 3.1 Introduction

This chapter describes the simulator designed for real time scheduler to schedule the real time task set and measure energy parameters. This chapter also describes the platform model, task model and energy model used for validating the existing and proposed algorithms and analyzing the system performance in terms of energy consumption. Various energy aware EDF versions are implemented, tested and analyzed with this simulator. The test bench for experimental evaluations are synthetically generated using standard benchmark suites. The computation of major energy components affecting the overall energy consumption while scheduling is also explained in this chapter.

### 3.2 SMART - A Simulator for MC-HRTS

A new framework named SMART - Simulator for MC hArd Real Time system is designed and implemented to find the schedule and measure energy parameters, idle and shutdown duration, number of scheduling decisions, number of preemptions and cache impacts. Various existing and proposed

algorithms are designed and implemented using SMART to produce the schedules. Various energy aware versions of EDF algorithm developed for various MC platforms using SMART are shown in table 3.1. Each scheduler is designed to execute in critical section so as to take the scheduling decision for a single core.

### **3.3 Platform Model**

SMART is designed for homogeneous symmetric MC systems that support multiple discrete voltage levels and operating frequencies. The experiment is conducted using 8 discrete voltage levels common to all the cores ranging between 0 volt to 1 volt for different states of core. For shutdown state, it is considered as zero. For idle state, 50% of maximum voltage is chosen and for the active state of core, the voltage levels chosen are 50%, 60%, 70%, 75%, 80%, 90% and 100% of maximum voltage. The energy components are considered for 70nm technology Transmeta Crusoe processor reported by [4]. According to [4], the maximum frequency at 1 volt is 3.1 GHz with 0.43 nF of capacitance. SMART is designed for the system that support multiple modes like shutdown, idle and active and system that allows migration of jobs between cores.

### **3.4 Task Model**

SMART is designed for CPU bound, periodic, independent, tasks with hard deadlines that do not share resources other than processor/core. SMART is

Table 3.1: Algorithms and Techniques used

No.	Algorithm	DVFS technique	DP technique	Migration Support
1	<b>MFFBP</b> (Modified First Fit Bin Packing)	-	-	-
2	<b>EDF</b> (EDF without shutdown)	No	No	No
3	<b>FFEDF_SD</b> (Earliest Deadline First with shutdown and FFBP)	No	No	No
4	<b>MFFEDF_SD</b> (Earliest Deadline First with shutdown and MFFBP)	No	No	No
5	<b>ccEDF</b> (cycle conserving EDF)	Yes	No	No
6	<b>ConvDPS</b> (Conventional Dynamic Procrastination Scheduler)	No	Yes	No
7	<b>DPS</b> (Dynamic Procrastination Scheduler)	No	Yes	No
8	<b>DPVFS</b> (Dynamic Procrastination cum Voltage/Frequency Scaling)	Yes	Yes	No
9	<b>OASIS</b> (Optimal stAtic energy Scheduler with mIgration and dynamic procraStination)	No	Yes	Yes
10	<b>HANDT</b> (Hare AND Tortoise scheduler)	Yes	Yes	Yes

---

designed to validate the existing and proposed scheduling algorithms with synthetically generated benchmark program suites. Similar to [4] [9] [10] [11], the experiment is conducted using several randomly generated task sets, each containing 20 tasks. Such randomly generated tasks are used as the common validation methodology in real time scheduling. Based on [4], tasks were assigned a random period between the range [250 ms, 8000 ms] and Worst Case Execution Time (WCET) between 35% and 80% of the period such that the total utilization of the task set varies from 140% to 305% in a set of 2, 3 and 4 cores. The experiments obtain the actual execution times (AET) using Gaussian distribution with mean,  $=(\text{WCET}+\text{AET})/2$  and standard deviation  $= (\text{WCET}-\text{AET})/\text{no of tasks}$ . The AET of the task is varied between 10% and 100% of its WCET in steps of 10%. All tasks are simulated to execute till hyper-period, i.e. least common multiple of periods of all tasks in the task set since the pattern of the schedule repeats after hyper-period. All these algorithms follow Modified First Fit (MFF) and First Fit(FF) Bin Packing approximation methods for task allocation.

## 3.5 Evaluation Parameters

The significant parameters measured as part of SMART simulator are described in this section.

1. **Hyperperiod:** The hyperperiod duration is computed as least common multiple of periods of all tasks. For an in-phase task set, the pattern of schedule repeats every hyperperiod. Thus the task set is scheduled for a hyperperiod duration. All parameters are analysed in this duration which guarantees successful completion of all the released jobs of all tasks.
2. **Number of scheduling decisions:** The scheduling decisions are the events when scheduler has to decide the next job to be executed by selecting the highest priority ready job from the queue and dispatching it to the core. These events are arrival of job and completion of job.
3. **Number of preemptions:** The arrival of job sometimes may result into preemption of running job. Each preemption results into context switching and cache impact overheads.
4. **Number of context switches:** The switching between two jobs causes time and energy consumption in changing the context. Apart from the context switching due to preemption, on job completion, if



the ready queue is non-empty there exist a context switch between completed job and next high priority job.

5. **Number of cache impacts:** Each context switch results into cache impact.
  
6. **Number of procrastination decisions:** On job completion, when the ready queue is empty, the schedulers decide upon procrastination. Some of these decisions may not result into procrastination as it depends on procrastination duration and shutdown threshold. In this case, the scheduler decides upon push migration and expands its computation for procrastination duration. This computation consumes more time and energy than the scheduling decisions on non-empty ready queue.
  
7. **Number of push migrations:** The push migration decisions which results into procrastination consumes time and energy in performing the push migration and finding the next scheduler invocation for cores involved in migration.
  
8. **Number of shutdown and wakeup decisions:** These are the events when core is shutdown. This consume energy in shutting down and waking up the core.

9. **Total shutdown duration (SD):** The duration for which the core remains in shutdown state without any energy consumption. It is used for finding the amount of energy saved over the shutdown duration produced by other algorithms.
  
10. **Number of idle decisions:** When the procrastination decision do not result into shutdown, the scheduler decide upon pull migration and expands its computation for set of pullable jobs.
  
11. **Number of pull migrations:** The non-empty set of pullable jobs results in performing the pull migration and finding the next scheduler invocation for cores involved in migration which consumes time and energy.
  
12. **Total idle duration (ID):** The duration for which the core remains in idle state with minimum energy consumption. It is used for finding the amount of energy saved over the idle duration produced by other algorithms.
  
13. **Number of voltage and frequency scaling decisions:** On job completion, when the ready queue is non-empty, the scheduler decides upon voltage/frequency scaling at which the next high priority job can be executed. This decision depends upon the slack available in the core. Some of these decisions may not result in reducing the

voltage/frequency due to less slack availability. In this case the jobs are executed at maximum voltage/frequency.

14. **Number of times the voltage/frequency are scaled:** The voltage/frequency scaling decisions which results in saving the energy by reducing the voltage/frequency causes the transition between voltage levels which consumes time and energy.
15. **Total Inactive Duration:** Inactive duration is the time when the core is in idle or shutdown state. It is used in finding the amount of unproductive energy consumed by the core.
16. **Total Active Duration:** When the core is in active state it consumes static and dynamic energy. This includes the scaled active duration whenever the jobs are executed at scaled voltage and frequency.
17. **Percentage of total shutdown duration over inactive duration:** To find how much duration of idle period is converted into shutdown, percentage of total shutdown over inactive is computed as  $(SD/(SD + ID))$ .

## 3.6 Energy Model

The energy model used to compute the energy consumption during scheduling is described in this section. We use the energy model and the technology parameters described in [4] [19]. The total energy consumption during execution is measured by considering the following energy components while executing with various scheduling algorithms:

### 1. Total static energy ( $E_{tot\_static}$ )

The major contributors of static energy are the sub-threshold leakage current ( $I_{sub}$ ) and the reverse bias junction current ( $I_{rev}$ ) which increases significantly with adaptive body bias voltage ( $V_{bs}$ ) [4]. The static energy consumption ( $E_{static}$ ) due to  $I_{sub}$  and  $I_{rev}$  is given by

$$E_{static} = V \cdot I_{sub} + V_{bs} \cdot I_{rev} \quad (3.1)$$

where  $V$  is the supply voltage.

According to [4],  $E_{static}$  is assumed to be 22 nJ per cycle. The total static energy  $E_{tot\_static}$  is computed as

$$E_{tot\_static} = total\ active\ duration * E_{static} \quad (3.2)$$

### 2. Total dynamic energy ( $E_{tot\_dynamic}$ )

Dynamic energy is the energy due to switching activity in a circuit. The energy consumption due to charging and discharging of gates on a CMOS core ( $E_{switch}$ ) and the short-circuit power consumption ( $E_{sc}$ )

contribute to dynamic energy consumption ( $E_{dynamic}$ ) [19]. It is given by

$$E_{dynamic} = E_{switch} + E_{sc} \quad (3.3)$$

where

$$E_{switch} = V^2 \cdot f \cdot C \quad (3.4)$$

where  $V$  is the supply voltage,  $f$  is the maximum frequency for job execution and  $C$  is the load capacitance.  $E_{sc} \propto V$

The total dynamic energy  $E_{tot\_dynamic}$  is computed as

$$E_{tot\_dynamic} = total\ active\ duration * E_{dynamic} \quad (3.5)$$

Based on [4], the experiments are performed at various voltage levels and frequencies, the dynamic energy is considered in the range 11 nJ to 44 nJ per cycle as shown in table 3.2.

Table 3.2: Dynamic energy

Voltage level (% of $V_{max}$ )	Dynamic energy (nJ)
50	11
60	15.84
70	21.56
75	24.75
80	28.16
90	35.64
100	44

### 3. Total core shutdown and wakeup energy ( $E_{tot\_psd}$ )

Core shutdown and wakeup energy is the energy due to flushing of

data cache during shutdown and memory accesses during wakeup. It is computed as the product of number of shutdown decisions ( $N$ ) and shutdown overhead ( $E_{sdo}$ ).

$$E_{tot\_psd} = N * E_{sdo} \quad (3.6)$$

The shutdown overhead is estimated by considering the on chip cache and other storage infrastructures. The cache size of the processor is assumed to be 32KB I-cache and 32KB D-cache. It is assumed that 20% lines of D-cache are dirty before shutdown which results in 6554 memory writes. By considering 13nJ of energy per memory write, the total energy requires for flushing the data cache is  $85\mu\text{J}$ . The energy and latency of saving the registers is assumed to be negligibly small. When a task resumes its execution, the locality of reference changes which causes cache misses. This additional cache misses is assumed to be 10% of the cache size in both I-cache and D-cache. This causes the total overhead of 6554 cache misses on wakeup. By considering 15nJ of energy per memory access, the total energy required for reading from memory and writing into cache is  $98\mu\text{J}$ . The energy required for updating TLBs and BTBs is assumed to be negligibly small. The energy required for charging the circuit logic is assumed to be  $300\mu\text{J}$ . Thus the total energy required to switch the processor between active and shutdown state is  $85 + 98 + 300 = 483\mu\text{J}$ .

Energy saved  $\geq 3$  times Energy consumed

i.e. (idle state energy + static energy) \* SDT  $\geq 3 * 483\mu\text{J}$

i.e. (22nJ + 22nJ) \* SDT  $\geq 1449\mu\text{J}$ . Thus SDT duration  $\geq 3659$  units.

#### 4. Total scheduler decision making energy ( $E_{tot\_sched}$ )

The priority driven scheduler is invoked on job arrival, core wakeup and job completion. The energy consumed for decision making at every scheduler invocation ( $E_{sched}$ ) is given by

$$E_{sched} = E_{decision} + E_{v/f} + E_{cio} + E_{job\_dispatch} \quad (3.7)$$

where  $E_{decision}$  is the energy requires for computing the procrastinated idle duration whenever the core is idle or for selecting the highest priority job from ready queue and allocating it to the core and/or checking for job(s) migration.

$E_{v/f}$  is the energy for voltage and frequency selection for task execution,  $E_{cio}$  is the cache impact overhead caused due to context switching after task completion or preemption and

$E_{job\_dispatch}$  is the energy required for dispatching the job.

It is assumed to be  $2\mu\text{J}$  energy for voltage and frequency selection,  $2\mu\text{J}$  for procrastination decisions,  $2\mu\text{J}$  for migration,  $98\mu\text{J}$  for cache impact and  $40\mu\text{J}$  for job dispatcher. According to [18], the feasible shutdown interval threshold considered is 2ms.

The total scheduler decision making energy ( $E_{tot\_sched}$ ) is computed as the product of total number of scheduler invocations ( $S$ ) and  $E_{sched}$ .

$$E_{tot\_sched} = S * E_{sched} \quad (3.8)$$

#### 5. Total idle duration energy ( $E_{tot\_idle}$ )

The core consumes static energy and minimum dynamic energy even

when it is idle. Thus the idle duration energy ( $E_{idle}$ ) is given by

$$E_{idle} = E_{static} + E_{dynamic} \quad (3.9)$$

$E_{static}$  for idle duration is considered same as that for active duration, i.e. 22nJ.  $E_{dynamic}$  for idle duration is computed using minimum voltage. Thus  $E_{dynamic}$  is considered as 11nJ.

The total idle duration energy  $E_{tot\_idle}$  is computed as.

$$E_{tot\_idle} = total\ idle\ duration * E_{idle} \quad (3.10)$$

Thus the **total energy consumption** ( $E_{total}$ ) is computed as

$$E_{total} = E_{tot\_stat} + E_{tot\_dyn} + E_{tot\_psd} + E_{tot\_sched} + E_{tot\_idle} \quad (3.11)$$

## 3.7 Summary

The system model used for development and experimentation of the existing and proposed algorithms is explained in this chapter. This includes the description of SMART simulator, platform model, task model and energy model. The computation of energy components affecting overall energy consumption is also explained.



# Chapter 4

## Energy Efficient Scheduling in MC-HRTS

### 4.1 Introduction

This chapter focuses on energy efficient scheduling for MC Hard Real Time System at task allocation and scheduling phases. It explains the proposed task allocation method - Modified First Fit Bin Packing (MFFBP). MFFBP is the variant of conventional Bin Packing approximation method - First Fit Bin Packing (FFBP) used for task allocation. This chapter also elaborates the design of two dynamic schedulers - Dynamic Procrastination Scheduler (DPS) and Dynamic Procrastination cum Voltage/Frequency Scaling (DPVFS) scheduler. In DPS, MFFBP and dynamic procrastination techniques are combined to reduce static and dynamic energy consumptions of the schedule. DPS is applicable for MC-HRTS supporting shutdown. DPS is extended to DPVFS for the systems supporting discrete operational voltages and frequencies to further reduce the dynamic energy consumption by combining it with DVFS technique. In this chapter, the schedulers designed using DPS and DPVFS (with FFBP and MFFBP as allocation) are compared with schedules of NO procrastination (MFFNOPRO), static procrastination

(MFFSTATICPRO) and Cycle Conserving Earliest Deadline First (ccEDF) to evaluate the performance and energy consumption.

## 4.2 Task Allocation for Energy Minimization

Widely used task allocation methods are on-line, semi-on-line and off-line [41]. On-line and semi-on-line methods are suitable for tasks having sporadic release time that can be scheduled with existing task set without missing deadlines. Off-line methods are best suited for fixed task set having temporal parameters known a priori. Unlike on-line and semi-on-line, off-line method gives scope of rearranging the task set. This work considers off-line allocation as it is dealing with task set having only periodic tasks with implicit deadlines. Bin Packing approximation is one of the most efficient off-line task allocation method used in MC systems [9]. Widely used polynomial-time Bin Packing approximation algorithms are FFBP, Best-Fit, Next-Fit and Worst-Fit [32] [34] [41]. Amongst these FFBP has the best performance ratio of  $11/9$  [41]. FFBP arranges the tasks in non increasing order of utilizations and finds the optimal number of cores required for the given task set [32] [41]. Several variants of these algorithms are designed like Best-K-fit, First-Fit Decreasing, Best-Fit Decreasing etc [41]. The arrangement of tasks based on utilization gives optimal number of cores but does not help in increasing shutdown duration of the schedule. This work aims at increasing the shutdown duration and optimizing energy consumption of the schedule. This is done by a new allocation algorithm - Modified FFBP (MFFBP).

### 4.2.1 MFFBP Algorithm

Task allocation of MFFBP is explained in Algorithm [1].

The data structures used in the algorithms are:

- $T[]$  - global task set,
- $C[]$  - set of cores,
- $S_i[]$  - sub taskset allocated to core  $C_i$ .

The abbreviations used in the algorithms are:

- util - utilization
- remutil - remaining utilization

MFFBP works in two phases - sorting and packing. MFFBP arranges the tasks in non-decreasing order of their periods. Then it allocates a new task  $T_i$  to the first (lowest indexed) core  $C_j$  into which it can fit. Thus if  $C_j$  is partially filled with utilization  $U_{C_j}$ ,  $T_i$  can be allocated to  $C_j$  iff  $U_{C_j} + U_{T_i} \leq 1$ . Otherwise  $T_i$  is allocated to a new core. Thus lower period tasks are allocated to same set of cores which cannot be considered for shutdown as maximum shutdown possible for core is  $2 * P_{low} - 2 * E_{low}$  where  $P_{low}$  and  $E_{low}$  are the period and WCET of lowest period task in the task set respectively. The cores having higher period tasks will be able to shutdown for a longer duration depending on the AETs of the jobs at run time.

---

**Algorithm 1: MFFBP**

---

**Input:** Task set  $T[T_1, T_2, \dots, T_N]$  arranged in non-decreasing order of periods, set of cores  $C[C_1, C_2, \dots, C_M]$

**Output:**  $P$ =Minimum no. of cores,  $\forall_{i=1 \text{ to } P}$  sub taskset  $S_i[]$

```

1 P=1
2 foreach task  $T_i \in T[]$  do
3   flag=0
4   foreach  $k = 1 \text{ to } P$  do
5     if  $(C_k.\text{remutil} \geq T_i.\text{util})$  then
6        $S_k[] \leftarrow T_i$ 
7        $C_k.\text{remutil} -= T_i.\text{util}$ 
8       flag=1 and break
9     end
10  end
11  if (flag is 0) then
12    P=P+1
13    if  $(P > M)$  then
14      Error: Insufficient cores and Exit
15     $S_P[] \leftarrow T_i$ 
16     $P.\text{remutil} -= T_i.\text{util}$ 
17  end
18 end
19 return  $P, \forall_{i=1 \text{ to } P}$  sub taskset  $S_i[]$ 

```

---

Table 4.1: Task set 1

Taskno	Period	WCET
$T_0$	40	9.4
$T_1$	60	15
$T_2$	50	20
$T_3$	80	19
$T_4$	100	20
$T_5$	140	25
$T_6$	120	20

### 4.2.2 Motivating Example

Consider a task set consisting of seven periodic hard real-time tasks with temporal parameters shown in table 4.1. In FFBP, the tasks are first arranged based on non-increasing order of their utilization. The resultant order is FFT =  $\{ T_2, T_1, T_3, T_0, T_4, T_5, T_6 \}$ . The tasks in FFT are allocated to different cores based on FFBP algorithm. In this example task set, the minimum number of cores is 2 and the sub task sets are FS<sub>1</sub> =  $\{ T_2, T_1, T_3 \}$  and FS<sub>2</sub> =  $\{ T_0, T_4, T_5, T_6 \}$  allocated to cores  $C_1$  and  $C_2$  respectively. The task set in MFFBP is first arranged based on non-decreasing order of their periods. The resultant order is MFFT =  $\{ T_0, T_2, T_1, T_3, T_4, T_6, T_5 \}$ . The tasks in MFFT are allocated to different cores based on MFFBP algorithm. The task allocation in MFFBP guarantees that the higher period tasks are allocated to the last core. In this example task set, the minimum number of cores is 2 and the sub task sets are MS<sub>1</sub> =  $\{ T_0, T_2, T_1 \}$  and MS<sub>2</sub> =  $\{ T_3, T_4, T_6, T_5 \}$  allocated to cores  $C_1$  and  $C_2$  respectively. To have uniformity in analysis, the utilization of FS<sub>2</sub> is made almost same as MS<sub>2</sub> which is approximately 78%. In case of FFBP, maximum shutdown durations possible for cores  $C_1$  and  $C_2$

are 60 and 61.2 respectively where as in MFFBP, it is 60 and 122 respectively. Assuming shutdown threshold of 100, cores  $C_1$  and  $C_2$  can never be shutdown in FFBP where as core  $C_2$  can be shutdown in MFFBP.

### 4.2.3 Analysis of MFFBP algorithm

This section describes schedulability and complexity analysis of MFFBP algorithm.

#### Schedulability analysis

**Theorem 4.2.1** *Any periodic task set  $T$  with  $N$  tasks having implicit deadlines can be feasibly scheduled on  $M$  identical cores by dynamic priority scheduling algorithm on satisfying the following condition [76].*

$$U_{tot} \leq M(1 - U_{max}) + U_{max} \quad (4.1)$$

where  $U_{tot} = \sum_{i=1}^N U_i$ ,  $U_{max} = MAX(T_i.WCET/T_i.P)$ ,  $T_i \in T$ ,  $i = 1$  to  $N$

**Proof:** Since the MC system follows static task allocation for a periodic task set, schedulable utilization of the task set can be found using equation 4.1. It gives the sufficient schedulability condition for a system containing  $M$  identical cores, each scheduled on dynamic priority basis.

**Theorem 4.2.2** *Any periodic task set  $T$  with  $N$  tasks having implicit deadlines satisfying Equation 4.1 can be feasibly allocated to at most  $M$  identical cores, each executing a dynamic priority scheduling algorithm, using MFFBP allocation method such that the overall shutdown duration is high.*

**Proof:** MFFBP arranges the tasks in non-decreasing order of periods and allocates the lower period tasks to initial K cores as first set of cores and higher period tasks to remaining K+1 to M cores as second set of cores.

$$U_{tot} = \sum_{i=1}^K U_i(P_{low}) + \sum_{j=K+1}^M U_j(P_{high}) \quad (4.2)$$

Equation 4.2 shows this allocation does not allow first set to shutdown but increases the probability of shutdown in second set of cores. Thus the overall shutdown duration is high in MFFBP. Since MFFBP considers task level dynamic priority scheduling algorithms with 100% utilization in each core, it maintains the schedulability condition of each core after task allocation.

**Complexity analysis:** MFFBP involves two steps. First step is to arrange the tasks in non-decreasing order of period which takes  $\mathcal{O}(N \log N)$  time where N is total number of tasks. Second step is to allocate the next task to the first possible core by scanning the cores in the order  $C_1, C_2, C_3, \dots, C_M$ . If a new core is needed, it increments the count of number of cores. The first task requires a scan of  $C_1$  only. Second task requires scanning at most  $C_1$  and  $C_2$ , third task scans at most  $C_1, C_2$  and  $C_3$ ; etc. Thus the total number of scans takes  $\mathcal{O}(N * M)$  time. The run time complexity of MFFBP algorithm is  $\text{Max} \{ \mathcal{O}(N \log N), \mathcal{O}(NM) \} = \mathcal{O}(N \log N)$  as  $\log N$  is larger than M due to limited number of cores.

#### 4.2.4 Experimental evaluation

**Experimental setup:** The experimentation is conducted using the task model described in Chapter 3. A new framework named Multi-Core Bin Packing (MCBP) is designed and implemented to measure the performance of MFFBP task allocation with EDF scheduling algorithm in comparison with seminal algorithms. The task sets are allocated using FF and MFF Bin Packing methods. They are scheduled using EDF with shutting down the core whenever the idle duration is more than the shutdown threshold (EDF\_SD). The framework evaluates FFEDF\_SD and MFFEDF\_SD scheduling algorithms based on shutdown duration, static and total energy consumptions.

**Experimental results:** Let  $S_{FFEDF\_SD}$  and  $S_{MFFEDF\_SD}$  be the schedules using FFEDF\_SD and MFFEDF\_SD schedulers respectively.

Figure 4.1 shows percentage of active, idle and shutdown period in  $S_{FFEDF\_SD}$  and  $S_{MFFEDF\_SD}$  for utilizations 210%, 240%, 295% and 305% for 3, 3, 4, 4 cores respectively. On an average, MFFEDF\_SD produces 183% of more shutdown period than FFEDF\_SD schedule.

Figure 4.2 shows the static energy consumption for different utilizations. MFFEDF\_SD consumes less static energy compared to FFEDF\_SD. This is because the static energy is inversely proportional to the shutdown duration. On an average, MFFEDF\_SD reduces the static energy by 13.43% over FFEDF\_SD algorithm.



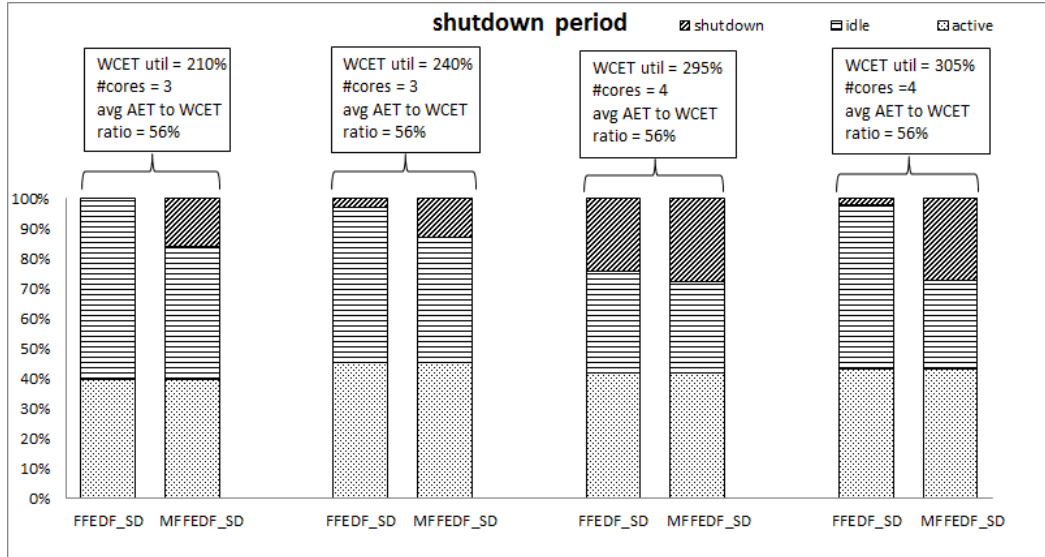


Figure 4.1: Percentage of active, idle and shutdown period for varying utilizations

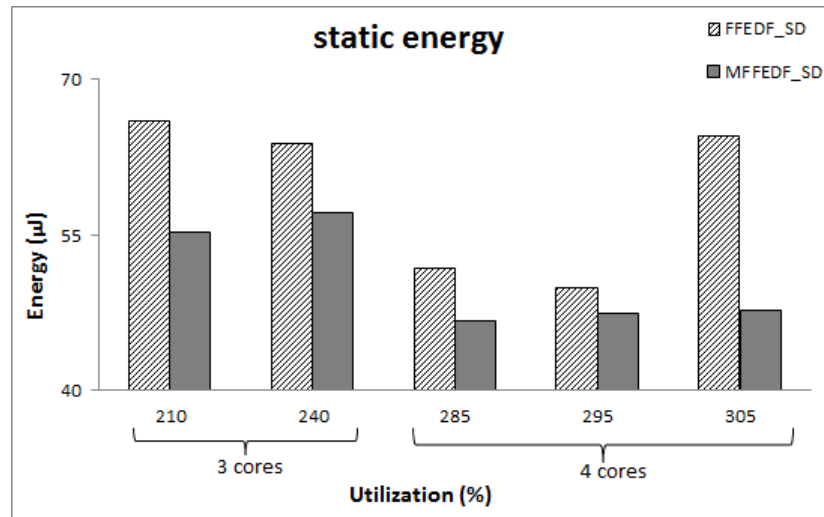


Figure 4.2: Static energy consumption per unit for different utilizations

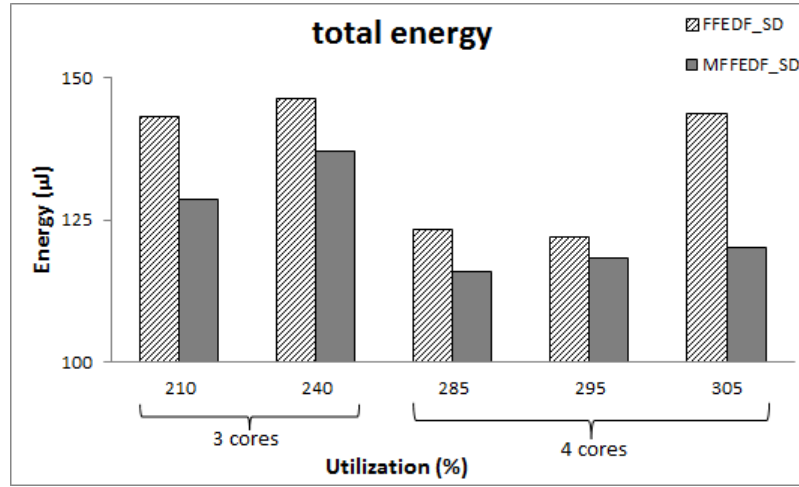


Figure 4.3: Total energy consumption per unit for different utilizations

Figure 4.3 shows the total energy consumption for different utilizations. MFFEDF\_SD consumes less total energy compared to FFEDF\_SD as it consumes less static energy. On an average, MFFEDF\_SD reduces the total energy by 8.42% over FFEDF\_SD algorithm.

### 4.3 Energy Efficient Dynamic Schedulers

Once the tasks are allocated to the corresponding cores using allocation algorithms, they are scheduled independently on each core. As each instance of the task (job) has varied execution time and most of the jobs has lesser core utilization than worst case, we preferred to use dynamic schedulers. The deadline driven methods like EDF are more suitable to design energy efficient scheduler over RM because of its high schedulability and over Least Laxity First (LLF) because of its reduced complexity. This section describes two novel energy aware schedulers - DPS and DPVFS. These schedulers (i) select

the highest priority job and dispatches it to the core for execution, (ii) find the corresponding voltage/frequency for its execution and (iii) decide the state of core to be active, idle or shutdown.

### 4.3.1 Dynamic Procrastination Scheduler

In systems which is not supporting power down, the cores remain in idle state whenever they are not active. During this period, the core consumes both static and dynamic energy for unproductive work. The unproductive energy consumption of the cores can be reduced by shutting it down if it supports power down when the idle duration is longer. Procrastination is one of the most efficient techniques to optimize the energy consumption during idle period. In procrastination, the task executions are delayed to increase the idle durations so as to convert it into a longer shutdown duration. Static and Dynamic variants are the most widely used procrastination techniques. Various off-line (static) and on-line (dynamic) procrastination solutions have been proposed under different application/device settings. Static Procrastination is based on precomputed procrastination intervals on considering their WCET. As most of the jobs complete before their WCET, static procrastination may not offer optimal solution. The slack left due to early completion, is utilized by dynamic methods to increase the idle duration. Dynamic Procrastination is more aggressive in terms of shutdown duration and energy saving. Dynamic Procrastination Scheduler (DPS) is designed to aggressively find idle duration and shut the core whenever it is beyond shutdown threshold.

#### 4.3.1.1 Working of Conventional Static and Dynamic Procrastination Scheduler

Consider the task set shown in table 4.1. Consider jobs of tasks in  $MS_2 = \{T_3, T_4, T_6, T_5\}$  formed in Section 4.2.2 allocated to core  $C_2$  and shutdown threshold (SDT) as 40 units. In static procrastination, the procrastination time of every job is precomputed and stored in a table data structure. The procrastination time of jobs  $\{J_{42}, J_{33}, J_{62}, J_{52}, J_{43}, J_{34}, J_{63}, J_{53}, J_{44}$  and  $J_{35}\}$  are computed as  $\{220, 276.5, 294.7, 346, 336.4, 344, 386, 494.7, 425.2, 421\}$  respectively using the portion of WCET of jobs executing between the arrival and deadline of particular job. For e.g. the procrastination time of job  $J_{33}$  is computed by considering the portion of execution of jobs  $\{J_{33}, J_{43}, J_{52}, J_{62}\}$  executing between the release time and deadline of  $J_{33}$  i.e between 240 and 320. Thus the procrastination time for  $J_{33} = 320 - (19 + 4 + 7.14 + 13.33) = 276.5$ .

Figure 4.4 (a) shows the resultant schedule using static procrastination with AET same as WCET. At time 187, when there is no job ready for execution, looking at the precomputed table, the upcoming job can be procrastinated till the smallest procrastination time i.e. 220. As this produces the slack of  $220 - 187 = 33$  units which is less than the SDT, the scheduler do not procrastinate the upcoming job. Thus the core remains idle for 13 units i.e. till 200. At 220 the upcoming job is procrastinated till smallest procrastination time i.e. 276.5 and the core is shutdown for 56.5 units.

Figure 4.4 (b) shows the resultant schedule using static procrastination with AET less than WCET. Consider AET of jobs  $\{J_{42}, J_{33}, J_{62}, J_{43}, J_{34}$  and  $J_{52}\}$  as 80% of estimated WCET i.e.  $\{16, 15.2, 16, 16, 15.2, 20\}$ . As  $J_{42}$  needs only

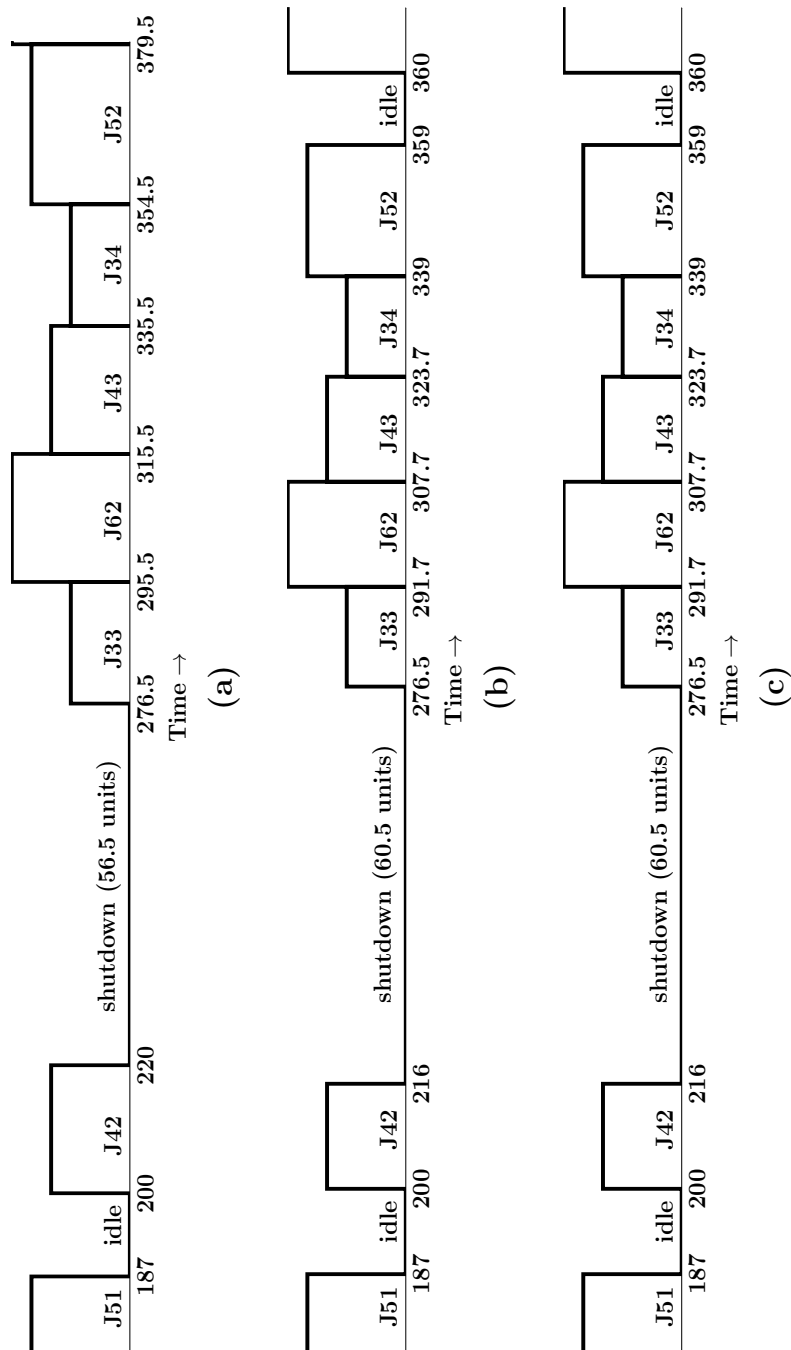


Figure 4.4: (a) Schedule using Static Procrastination with  $AET = WCET$  (b) Schedule using Static Procrastination with  $AET = 80\% WCET$  (c) Schedule using Conventional DPS

16 units instead of 20, the core becomes idle at 216. At 216, the upcoming job is procrastinated till 276.5 looking at the precomputed procrastinated time and the core is shutdown for 60.5 units. At 359, when there is no job ready for execution, looking at the precomputed table, the upcoming job can be procrastinated till the smallest procrastination time i.e. 386. As this produces the slack of  $386 - 359 = 27$  units which is less than the SDT, the scheduler do not procrastinate the upcoming job. Thus the core remains idle for 1 unit i.e. till 360.

Figure 4.4 (c) shows the resultant schedule using Conventional DPS (ConvDPS). In ConvDPS, when core becomes idle at 187, the procrastination duration is computed using portion of execution of jobs arriving before the deadline of highest priority job i.e.  $J_{33}$ . The jobs arriving between 240 and 320 are  $\{J_{33}, J_{62}, J_{52}, J_{43}\}$ . Thus the procrastination time =  $320 - (19 + 4 + 7.14 + 13.33) = 276.5$ . As the slack produced is more than SDT, the core is shutdown till 276.5 for 60.5 units.

#### 4.3.1.2 DPS algorithm

Like other priority driven schedulers, DPS also has job arrival and job completion as decision points. On each job arrival, the scheduler compares its priority with the executing job. The executing job is preempted if the newly arrived job has higher priority, otherwise the new job joins the ready queue. On job completion, if the ready queue is non empty, highest priority job is selected for execution. If ready queue is empty, DPS decides to procrastinate the job executions. DPS offers the maximum idle duration possible by taking

---

**Algorithm 2: DPS**

---

**Input:** time  $t$  when  $C_i.RQ$  is empty, sub taskset  $S_P[]$ **Output:** SIT,

TRUE for shutdown / FALSE otherwise

```

1  $J \leftarrow$  Job with earliest deadline( $D_{next1}$ ) after  $t$ 
2 if ( $D_{next1} - t - J.WCET < SDT$ ) then
3   |   SIT=next job arrival time
4   |   return  $SIT, FALSE$ 
5 end
6  $D_{next2} \leftarrow$  deadline of lowest priority job arriving before  $D_{next1}$ 
7  $L[] \leftarrow$  list of jobs releasing between  $t$  and  $D_{next2}$  arranged in
   |   non-increasing order of their absolute deadlines
8 SIT= $D_{next2}$ 
9 foreach job  $J_i \in L[]$  do
10  |   if ( $J_i.d > D_{next2}$ ) then
11  |   |   NSI -= ( $(D_{next2} - J_i.r) * (J_i.WCET / J_i.P) * U_p$ )
12  |   |   else
13  |   |   |   NSI -=  $J_i.WCET$ 
14  |   |   |   if ( $(J_i \neq lastjob) \ \&\&\ (SIT > J_{i+1}.d)$ ) then
15  |   |   |   |   SIT =  $J_{i+1}.d$ 
16  |   |   end
17 end
18 if ( $SIT - t < SDT$ ) then
19  |   SIT=next job arrival time
20  |   return  $SIT, FALSE$ 
21 end
22 return  $SIT, TRUE$ 

```

---

care of all job deadlines. In DPS, the Procrastinated Idle Duration (PID) is calculated by considering the executions of jobs arriving between current time  $t$  and  $D_{next}$ , where  $D_{next}$  is the deadline of the lowest priority job arriving before the nearest deadline. For all the jobs with deadline before  $D_{next}$ , WCET is considered, and for the jobs whose deadlines are after  $D_{next}$ , only a portion of their executions before  $D_{next}$  is considered. DPS decides whether to shutdown the core or keep it in idle state based on PID and shutdown threshold (SDT). If PID is more than SDT, the core is shutdown for Procrastinated Idle Duration after backing up the relevant data. Otherwise the core is kept idle until the next job arrival. DPS scheduler is explained in Algorithm [2].

#### 4.3.1.3 Motivating Example

Consider the task set shown in table 4.1. Consider jobs of tasks in  $MS_2 = \{T_3, T_4, T_6, T_5\}$  formed in Section 4.2.2 allocated to core  $C_2$  with AET same as estimated WCET. Figure 4.5 (a) shows the resultant schedule without procrastination. In EDF schedule without procrastination, at time 187, when there is no job ready for execution, the core is kept idle till next job arrival, i.e. till time 200. Figure 4.5 (b) shows the resultant schedule using DPS. In DPS, when core becomes idle at 187, the procrastination duration is computed. This is done by finding the job with earliest deadline, i.e. job  $J_{42}$  having WCET as 20 and deadline as 300. Assuming slack  $(300 - 187 - 20)$  is more than shutdown threshold (SDT), the algorithm decides to compute maximum procrastination duration. Job  $J_{62}$  with deadline 420 is the lowest priority job arriving before 300. Thus  $D_{next}$  is 420. List of jobs arriving between 200



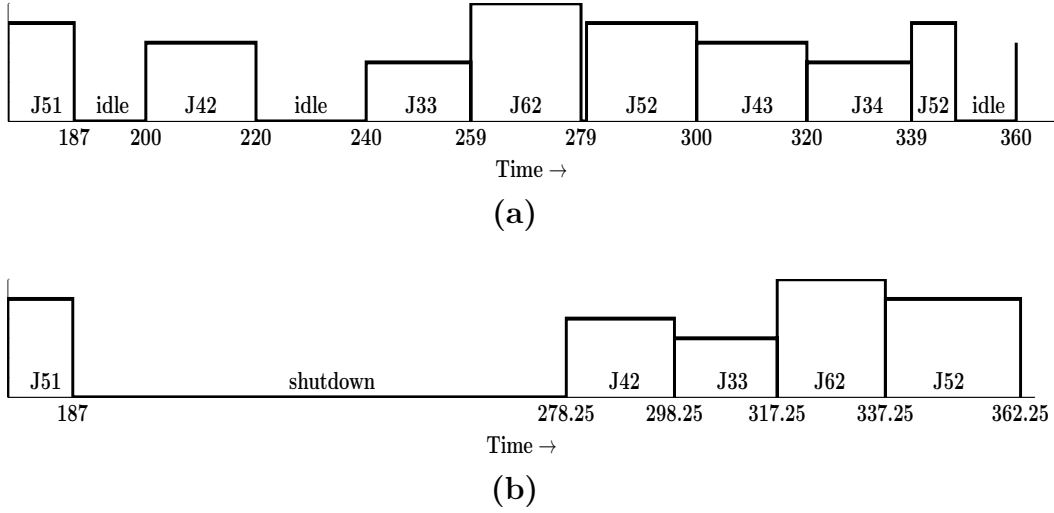


Figure 4.5: (a) Schedule without procrastination (b) Schedule with DPS

and 420 sorted based on deadline are  $\{ J_{44}, J_{53}, J_{35}, J_{62}, J_{43}, J_{34}, J_{52}, J_{33}$  and  $J_{42} \}$ . By iterating through all these jobs, DPS finds the latest time to start executing next job as 278.25. As slack  $(278.25 - 187)$  is much higher than SDT, the scheduler decides to shutdown core  $C_2$  till 278.25.

#### 4.3.1.4 Analysis of DPS Algorithm

This section describes schedulability, correctness and complexity analysis of DPS algorithm.

### Schedulability analysis

**Theorem 4.3.1** *Any periodic task set  $T_i$  with implicit deadlines having less than or equal to 100% utilization is DPS schedulable on a core  $C$ .*

**Proof:** DPS is a task level dynamic priority scheduling algorithm for individual core with decision points as job arrivals, job completions and

wakeup. For a given task set  $T_i$  on core  $C$ , DPS offers a valid schedule if it satisfies the utilization bound given in Equation 4.3.

$$U_C = \sum_{i=1}^N U_i \leq 1 \quad (4.3)$$

where  $U_i = Utilization\ of\ T_i \in T$

The procrastination decisions in DPS make sure it maintains the schedulability bound. Consider Figure 4.6. Let AP and PI denote Active Period and Procrastination Interval respectively. The shutdown decision is made at time

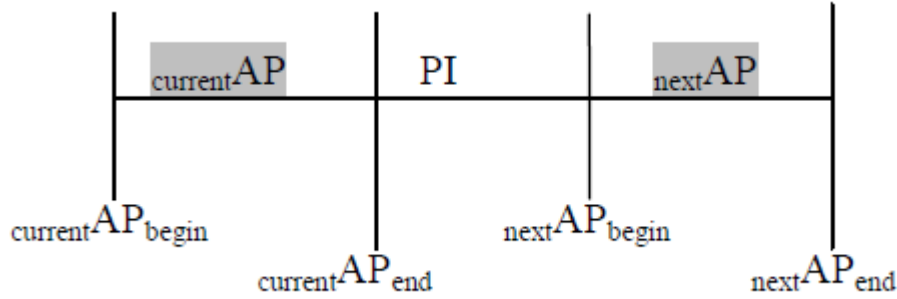


Figure 4.6: AP and PI

$t$  when the ready queue is empty, i.e. at the end of AP ( $currentAP_{end}$ ). It is decided to shutdown only if the PI is more than the shutdown threshold. Assume that all the jobs in  $nextAP$  are arranged in chronological order such that job  $j_i$  has higher priority than job  $j_j$ , job  $j_j$  has higher priority than job  $j_k$  and so on. At time  $t$ , procrastination duration  $Z_k$  of job  $j_k$  (job with earliest deadline after  $t$ ) is computed using the following equation:

$$j_k.WCET + Z_k + X + Y \leq PI + nextAP \quad (4.4)$$

where

$$X = \sum_{\forall j_i \in J_i: i \neq k \& j_i.r, j_i.d \leq D} j_i.WCET \quad (4.5)$$

$$Y = \sum_{\forall j_l \in J_l: l \neq k \& j_l.r \leq D, j_l.d > D} D - j_l.r \quad (4.6)$$

$D$  is the absolute deadline of job having release time  $j_j.r < j_k.d$  and deadline  $j_j.d > j_k.d$ ,

$J_i$  is the set of jobs in next active period ( $_{next}AP$ ) released before  $D$  and having deadline before  $D$ ,

$J_l$  is the set of jobs in  $_{next}AP$  released before  $D$  and having deadline after  $D$ .

On procrastination, all the tasks in  $_{next}AP$  will be executed with full frequency. Equation 5.5 shows that no job will miss its deadline due to postponement of job execution and the task set remains schedulable with procrastination.

## Correctness analysis

**Theorem 4.3.2** *The overall shutdown duration produced by the DPS scheduler is greater than or equal to the one produced by EDF scheduler.*

**Proof:** Let  $S_{DPS}$  and  $S_{EDF}$  be the schedules produced by DPS and EDF scheduling algorithms respectively. Let  $SD_{DPS}$  and  $SD_{EDF}$  be the total shutdown duration in  $S_{DPS}$  and  $S_{EDF}$  respectively. EDF keeps the core in shutdown state till next job arrival if the idle duration is more than the shutdown threshold. DPS algorithm procrastinates the execution of future jobs and merges the intermediate idle intervals within same core which results

into longer shutdown duration. Thus

$$SD_{DPS} \geq SD_{EDF} \quad (4.7)$$

**Theorem 4.3.3** *The schedule produced by the DPS scheduler offers better energy saving compared to EDF scheduler with shutdown.*

**Proof:** Let  $ID_{DPS}$  and  $ID_{EDF}$  be the total idle duration in  $S_{DPS}$  and  $S_{EDF}$  respectively. The objective of DPS to keep the core in shutdown state instead of idle to produce overall less idle duration compared to EDF. As active duration is constant in both the approaches,

$$SD_{DPS} + ID_{DPS} = SD_{EDF} + ID_{EDF} \quad (4.8)$$

Eq.(4.8)  $\implies$

$$total\_shut\_down\_duration \propto \frac{1}{total\_idle\_duration} \quad (4.9)$$

Eqs.(4.7) and (4.9)  $\implies$

$$ID_{DPS} \leq ID_{EDF} \quad (4.10)$$

Let  $SE_{DPS}$ ,  $SE_{EDF}$  be the static energy consumed and  $DE_{DPS}$ ,  $DE_{EDF}$  be the dynamic energy consumed while scheduling  $S_{DPS}$  and  $S_{EDF}$  respectively.

Eq.(4.7) and (4.10)  $\implies$

$$SE_{DPS} \leq SE_{EDF} \quad (4.11)$$

and

$$DE_{DPS} \leq DE_{EDF} \quad (4.12)$$

Let  $NSD_{DPS}$  and  $NSD_{EDF}$  be the number of shutdowns,  $SDE_{DPS}$  and  $SDE_{EDF}$  be the shutdown and wakeup energy in  $S_{DPS}$  and  $S_{EDF}$  respectively. The prolonged shutdown duration in DPS gives less number of shutdown intervals compared to EDF. Thus Eq.(4.7)  $\implies$

$$NSD_{DPS} \leq NSD_{EDF} \quad (4.13)$$

Eq.(4.13)  $\implies$

$$SDE_{DPS} \leq SDE_{EDF} \quad (4.14)$$

Let  $ENERGY_{DPS}$  and  $ENERGY_{EDF}$  be the overall energy while scheduling  $S_{DPS}$  and  $S_{EDF}$  respectively.

Eqs.(4.11), (4.12) and (4.14)  $\implies$

$$ENERGY_{DPS} \leq ENERGY_{EDF} \quad (4.15)$$

Thus the overall energy consumption with DPS scheduling algorithm is less in comparison with EDF approach.

## Complexity analysis

Complexity of maintaining priority queue is  $\mathcal{O}(N \log N)$  where  $N$  is the number of tasks allocated to the core. Selecting next job to run from a priority queue is a constant ( $C_1$ ) time operation. Preemption includes backing-up/restoring the critical data. Though preemption is considered as a major parameter, in experimental analysis it is considered as a constant ( $C_2$ ) time operation for complexity analysis. Thus the run time complexity of scheduler on job(s)

arrival is  $N \log N + C_1 + C_2$ . In asymptotic notation it is  $\mathcal{O}(N \log N)$ . The computation time complexity of PID is  $\mathcal{O}(K)$  where  $K$  is the number of jobs arriving between  $t$  and  $D_{next}$ . Transition from active to shutdown and vice versa includes backing-up/restoring the critical data. This is considered as constant ( $C_2$ ) time operation as explained above. Thus the run time complexity of scheduler on job completion when ready queue is empty is  $\mathcal{O}(K) + C_2$  and  $C_1$  when ready queue is non empty. In asymptotic notation it is  $\mathcal{O}(K)$ . Thus the run time complexity of DPS is  $\text{Max}\{\mathcal{O}(N \log N), \mathcal{O}(K)\} = \mathcal{O}(N \log N)$ .

#### 4.3.1.5 Experimental Evaluation

**Experimental setup:** The experimentation is conducted using the task model described in Chapter 3. The task sets are allocated using MFFBP method described in Section 4.2.1. The framework SMART described in Chapter 3 is used to find the schedule and measure the performance of DPS algorithm in comparison with seminal algorithms. The task sets are scheduled using MFFBP with Static Procrastination (MFFSTATICPRO), FFBP with Conventional DPS (FFConvDPS), FFBP with DPS (FFDPS), and MFFBP with DPS (MFFDPS) algorithms. The framework evaluates these scheduling algorithms based on shutdown duration, procrastination decision points, static, dynamic and total energy consumptions.

**Experimental results:** Figures 4.7, 4.8 and 4.9 show the effect of dynamic procrastination over static. Let  $S_{MFFSTATICPRO}$  and  $S_{MFFDPS}$  be the schedules using MFFSTATICPRO and MFFDPS schedulers respectively.

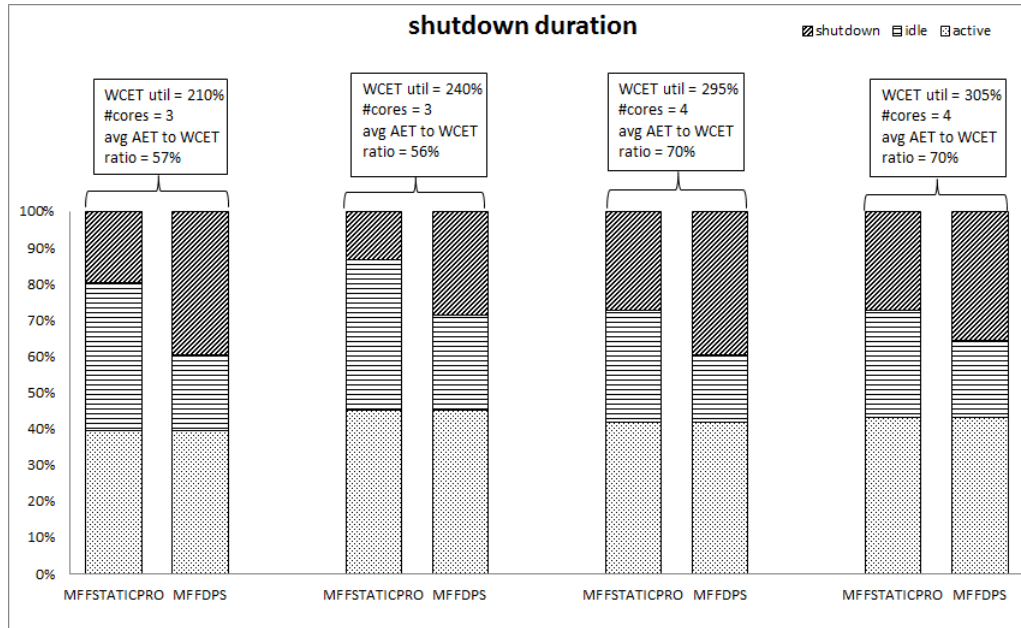


Figure 4.7: Percentage of active, idle and shutdown period for varying utilizations

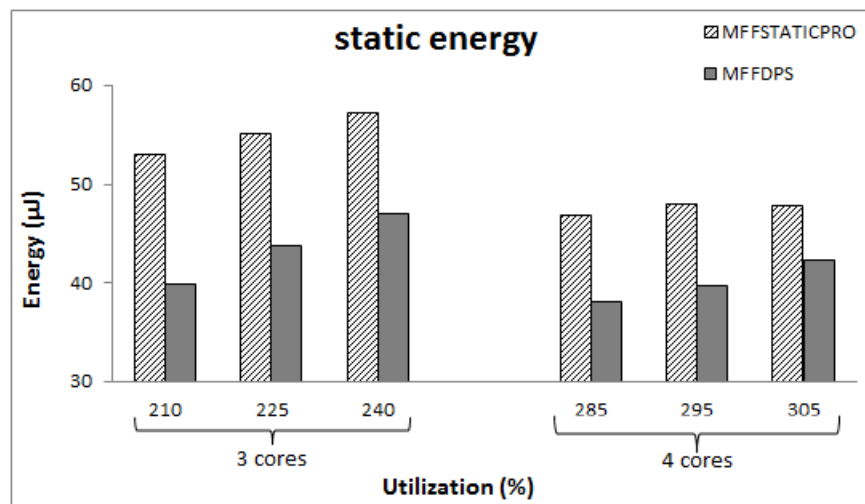


Figure 4.8: Static energy consumption per unit for different utilizations

Figure 4.7 shows percentage of active, idle and shutdown period in  $S_{MFFSTATICPRO}$  and  $S_{MFFDPS}$  for utilizations 210%, 240% with 3 cores and 295%, 305% with 4 cores. MFFDPS outperforms MFFSTATICPRO because of the difference in WCET and AET. On an average, MFFDPS produces 63.42% more shutdown period than MFFSTATICPRO schedule.

Figure 4.8 shows the static energy consumption for different utilizations. MFFDPS consumes the least static energy compared to MFFSTATICPRO. This is because the static energy is inversely proportional to the shutdown duration. On an average, MFFDPS reduces the static energy by 18.3% over MFFSTATICPRO algorithm.

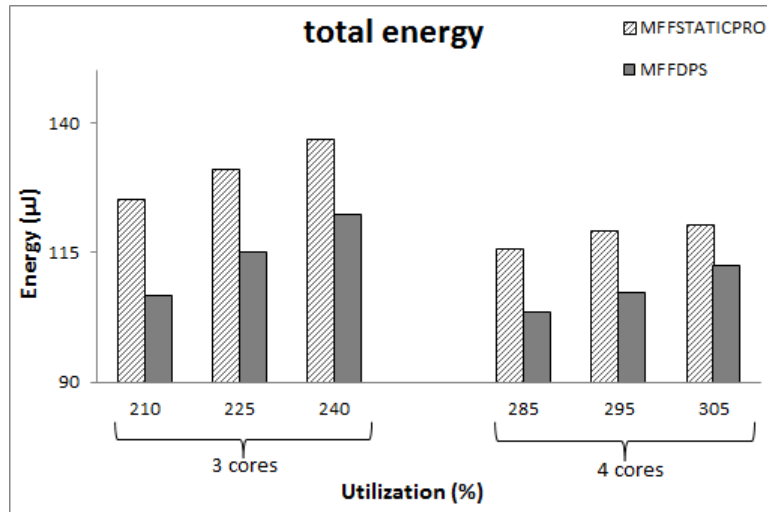


Figure 4.9: Total energy consumption per unit for different utilizations

Figure 4.9 shows the total energy consumption for different utilizations.



MFFDPS consumes the least static energy compared to MFFSTATICPRO. On an average, MFFDPS reduces the total energy by 10.7% over MFFSTATICPRO algorithm.

Figures 4.10, 4.11, 4.12, 4.13 and 4.14 show the effect of MFFBP and DPS methods over FFBP and Conventional DP methods. The experiments are carried out for utilizations 210%, 225%, 240% with 3 cores and 285%, 295%, 305% with 4 cores. Let  $S_{ConvDPS}$ ,  $S_{FFDPS}$  and  $S_{MFFDPS}$  be the schedules using Conventional Dynamic Procrastination, FFDPS and MFFDPS schedulers respectively.

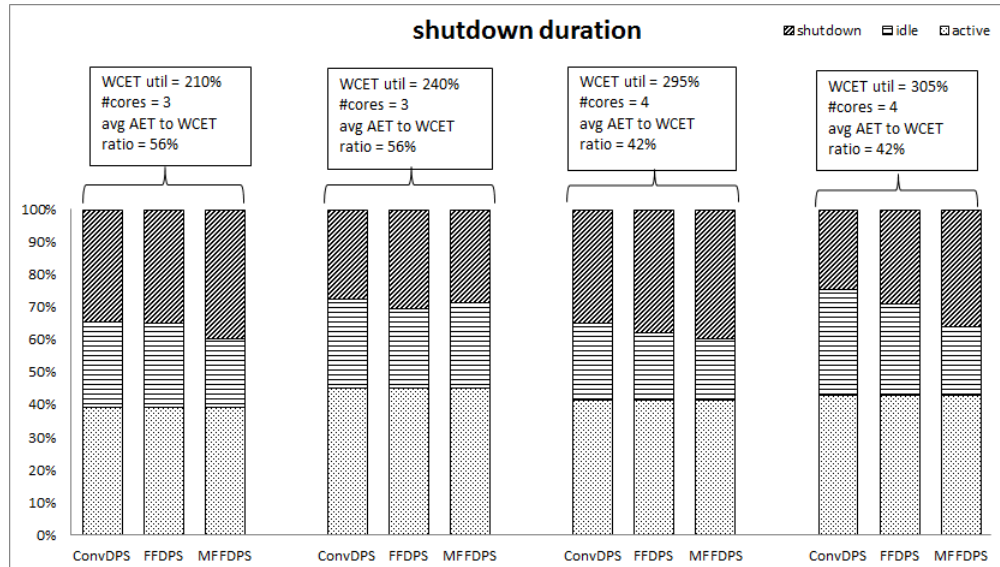


Figure 4.10: Percentage of active, idle and shutdown period for varying utilizations

Figure 4.10 shows percentage of active, idle and shutdown period in  $S_{ConvDPS}$ ,  $S_{FFDPS}$  and  $S_{MFFDPS}$ . On an average, MFFDPS produces 18.35% and 9.22% more shutdown duration than ConvDPS and FFDPS schedule respectively.

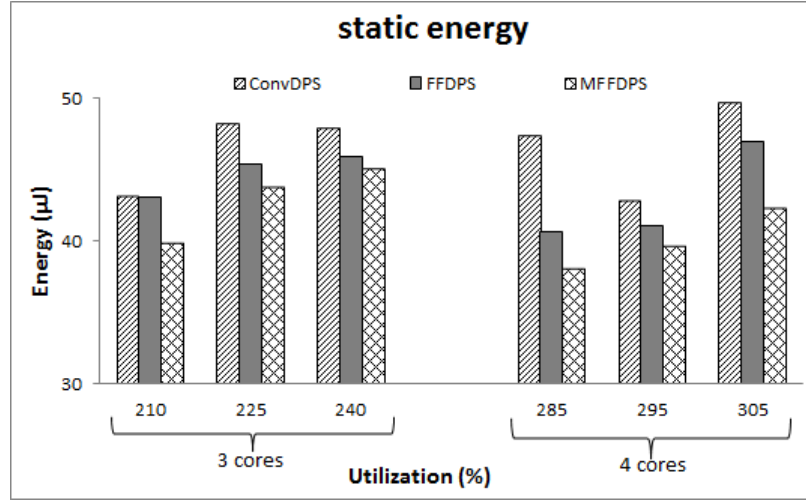


Figure 4.11: Static energy consumption per unit for different utilizations

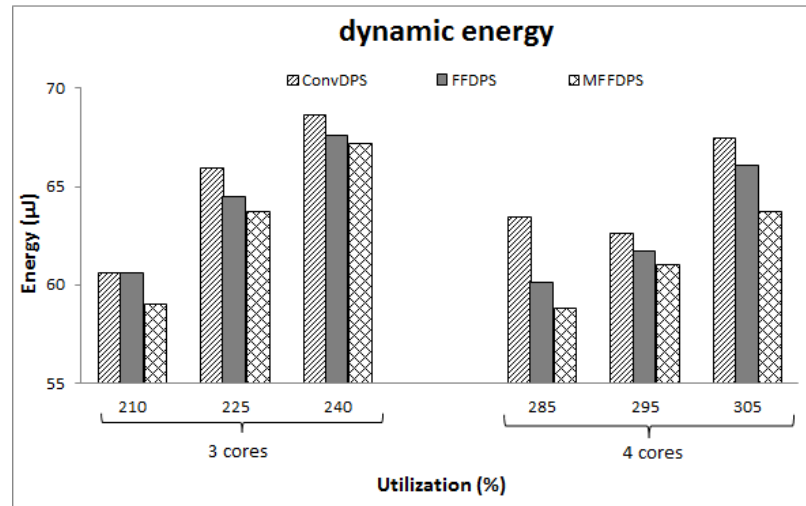


Figure 4.12: Dynamic energy consumption per unit for different utilizations

Figure 4.11 shows the static energy consumption for different utilizations. MFFDPS consumes the least static energy compared to ConvDPS and FFDPS. This is because the static energy is inversely proportional to the

shutdown duration. It is also observed that the static energy consumption increases with increase in utilization because of the shorter shutdown period. On an average, MFFDPS reduces the static energy by 10.7% and 5.33% over ConvDPS and FFDPS algorithms respectively.

Figure 4.12 shows the dynamic energy consumption for different utilizations. MFFDPS outperforms other algorithms as it produces more shutdown duration and less idle duration. On an average, MFFDPS reduces the dynamic energy by 4% and 1.8% over ConvDPS and FFDPS algorithms respectively.

Figure 4.13 shows the decision making energy consumption for different utilizations. In procrastination approach, when job queue is empty, energy is consumed in computing the procrastinated idle duration and taking a decision whether to procrastinate the job execution or keep the core idle. On an average, MFFDPS has 1.8% and 1.67% more decision making energy compared to ConvDPS and FFDPS algorithms respectively.

Figure 4.14 shows the total energy consumption for different utilizations. The total energy consumption follows the same trend as static and dynamic energy consumptions. The energy consumption increase with increase in utilization. This is mainly because of the static and dynamic energy components and less chance for shutting down in all procrastination based algorithms. Irrespective of utilizations, MFFDPS offers the least energy consumption followed by FFDPS and ConvDPS. On an average, MFFDPS

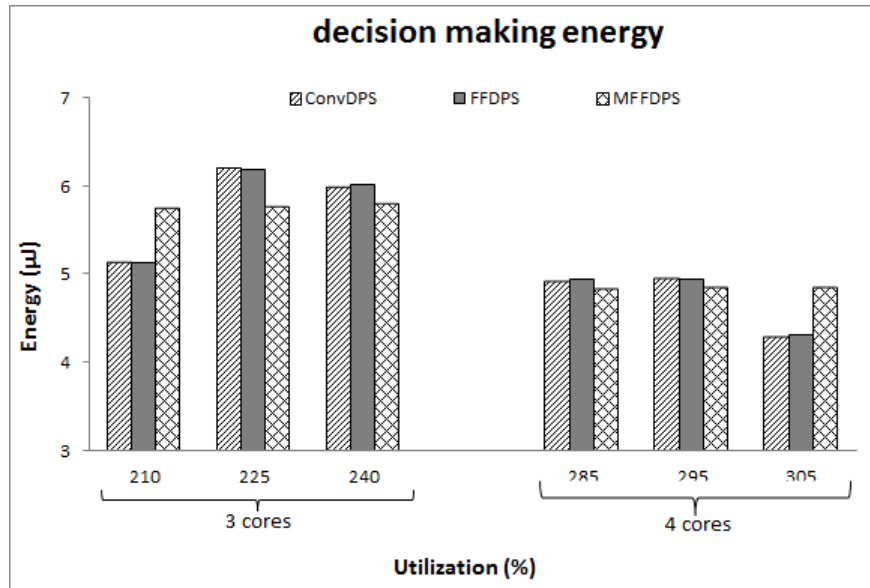


Figure 4.13: Decision making energy consumption per unit for different utilizations

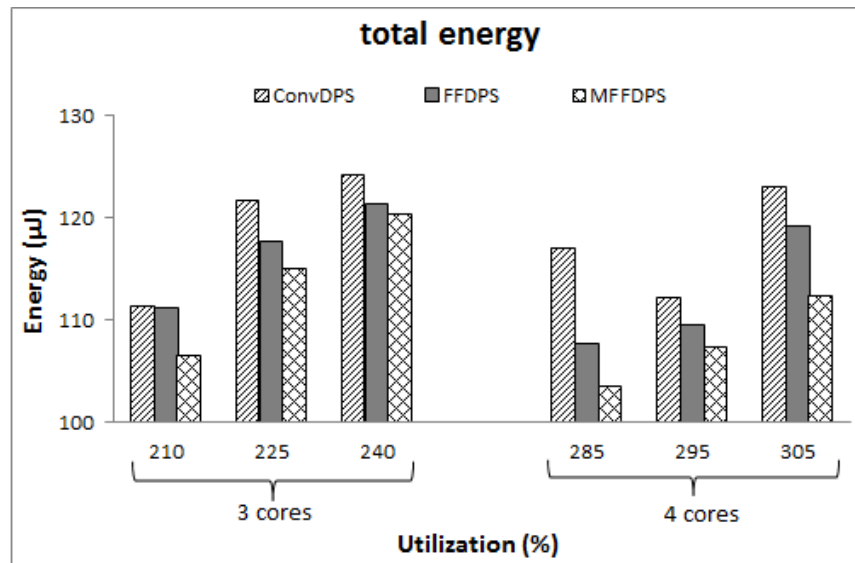


Figure 4.14: Total energy consumption per unit for different utilizations

reduces total energy consumption by 3.1% and 6.2% over FFDPS and ConvDPS algorithms respectively.

**Conclusion:** Along with the primary constraint of timeliness for real time tasks, MFFDPS scheduler also minimizes overall static and dynamic energy consumptions over MFFSTATICPRO, ConvDPS and FFDPS algorithms. It is applicable to the MC system supporting various power modes like shutdown, idle and active.

### 4.3.2 DPVFS Scheduler

The schedule produced by DPS may have some unused idle intervals consuming energy. These idle intervals can be converted into active by executing the jobs at reduced voltage and frequency. This is done using **Dynamic Procrastination cum Voltage and Frequency Scaling (DPVFS)** scheduler. DPVFS use DVFS technique to save dynamic energy and DP technique to save static and dynamic energy. It uses Cycle Conserving Earliest Deadline First (ccEDF) [5] method as DVFS technique and DPS method as DP technique. In DPVFS, before executing a job, the Next Idle Duration (NID) is computed using DPS. DPS helps in increasing the NID by converting the distributed short idle intervals to longer duration and finds the procrastinated idle duration (PID). Based on PID, DPVFS recommends a choice of slowing down the core or shutting down the core. If significant length of PID is available in future, the current job is executed at maximum voltage and frequency. Otherwise the scaled voltage and frequency for the job is computed and the job execution is extended for the available idle duration.

Thus the unused idle intervals are converted into active by executing the jobs at reduced voltage and frequency. DVFS incurs energy overhead due to voltage transitions and programming the clock frequency. Though these techniques work well independently, combining them to get optimized overall energy consumption without negatively affecting the performance of an application in MC system is still a challenge. This introduces challenges like optimal shutdown, optimal voltage/frequency scaling, trade off between static and dynamic energy consumption, preemption, cache impact overheads etc. To obtain optimal energy consumption, this work systematically incorporates DVFS and DP to balance the dynamic and static energy savings for the architectures that support multiple voltage and frequency levels and core shutdown.

#### 4.3.2.1 Working of ccEDF

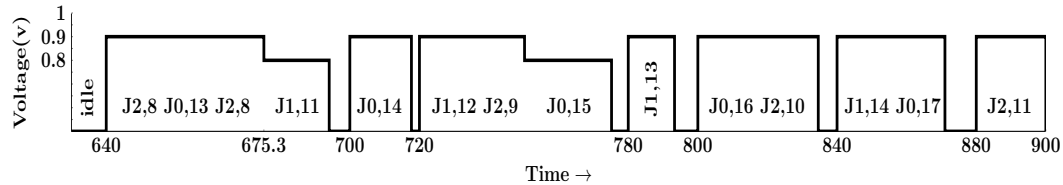


Figure 4.15: Schedule using ccEDF

Consider scheduling of jobs of tasks  $\{T_0, T_1, T_2\}$  with  $\{\text{Period, WCET}\}$  as  $\{50, 20\}$ ,  $\{60, 15\}$  and  $\{80, 19\}$  respectively. Figure 4.15 shows the resultant schedule using cycle-conserving EDF (ccEDF). Consider AET of jobs shown in figure as 80% of estimated WCET i.e.  $\{16, 12, 15.2\}$  for tasks  $T_0$ ,  $T_1$  and  $T_2$  respectively. In ccEDF, at each decision point i.e. on job arrival and on job completion, the scaled voltage is computed based on AET of finished jobs

and WCET of other jobs.

At time 640, when the core becomes idle, the scaled voltage is computed using WCET of all tasks i.e. 0.9v. The highest priority job  $J_{28}$  is executed at 0.9v and scaled frequency. On its completion, the scaled voltage is computed by considering its AET and WCET of other tasks i.e. 0.8v. The next highest priority job  $J_{1,11}$  is executed at 0.8v and scaled frequency. Similar scaled voltage is computed at each decision points.

#### 4.3.2.2 DPVFS algorithm

Algorithms [3] and [4] show SELECTOR and DPVFS algorithms respectively.

The data structures used in the algorithms are:

- $S_i[]$  - sub taskset allocated to core  $C_i$ ,
- $W[]$  - an array of all jobs of tasks in  $S_i$  ready at  $t$ .

The abbreviations used in the algorithms are:

- |                                |                            |
|--------------------------------|----------------------------|
| • WT - Wakeup Time             | • SV - Scaled Voltage      |
| • NJAT - Next Job Arrival Time | • SF - Scaled Frequency    |
| • AP - Active Period           | • NID - Next Idle Duration |
| • ID - Idle Duration           | • R - Running Job          |
| • SDT - ShutDown Threshold     |                            |

**Algorithm SELECTOR:** The SELECTOR algorithm takes task set allocated to the core as input and finds the active period (AP). If AP is zero, it sets  $C_i.SV$  to maximum. Otherwise it calls DPS algorithm to compute

---

**Algorithm 3: SELECTOR**


---

**Input:** sub taskset  $S_i[]$  allocated to core  $C_i$

**Output:**  $C_i.SV$

```

1 foreach job  $J_i \in W[]$  do
2   |  $C_i.AP = C_i.AP + J_i.WCET$ 
3   |  $W[] \leftarrow$  jobs arriving between t and  $C_i.AP$ 
4 end
5 if ( $C_i.AP == 0$ ) then
6   |  $C_i.SV = \text{MAX VOLTAGE}$ 
7  $C_i.NID = \text{DPS}(t+C_i.AP) - t$ 
8 if ( $C_i.NID < SDT$ ) then
9   |  $C_i.SV =$  nearest higher voltage than  $(C_i.AP / (C_i.AP + C_i.NID))$ 
10 return  $C_i.SV$ 

```

---

next idle duration (NID). DPS algorithm is described in Algorithm [2]. If NID is less than SDT, it computes the appropriate voltage and frequency using AP, NID and available voltage/frequency level and returns the scaled voltage for the core.

**Algorithm DPVFS:** The DPVFS algorithm takes task sets of all cores as input and generates the valid schedule with voltage scaling or procrastination. The task sets are allocated to the cores using Modified First Fit Bin Packing (MFFBP) described in Section (4.2). It schedules the jobs on following two events:

**Event 1: On job arrival:** On job arrival, it updates the execution time of running job, AP, NID and inserts new job in priority queue. If AP is



---

**Algorithm 4: DPVFS**

---

**Input:**  $\forall_{i=1 \text{ to } P}$  sub taskset  $S_i$ []  
**Output:** Schedule with voltage scaling or procrastination

- 1 **Event1: On Arrival of job J**
- 2 Update R.WCET,  $C_i$ .AP and  $C_i$ .NID with R.AET in SF
- 3 **if** ( $C_i$ .AP == 0) **then**
- 4 |     **SELECTOR**( $C_i$ )
- 5 |     Execute job J at  $C_i$ .SV;
- 6 **end**
- 7 **else**
- 8 |     Execute highest priority job at  $C_i$ .SV
- 9 **End Event1**
- 10 **Event2: On Completion of job J**
- 11 Update R.WCET,  $C_i$ .AP &  $C_i$ .NID with R.AET in SF
- 12 **if** ( $(C_i$ .AP == 0) && ( $C_i$ .NID == 0)) **then**
- 13 |     **SELECTOR** ( $C_i$ )
- 14 **if** ( $C_i$ .NID  $\geq$  SDT) **then**
- 15 |     **if** ( $C_i$ .RQ is empty) **then**
- 16 |     |     Keep core  $C_i$  in shutdown state till  $t+C_i$ .NID
- 17 |     |      $C_i$ .SV = MAX VOLTAGE
- 18 |     **end**
- 19 |     **else**
- 20 |     |     Execute highest priority job at MAX VOLTAGE
- 21 **end**
- 22 **else**
- 23 |     **if** ( $C_i$ .RQ is empty) **then**
- 24 |     |     Keep core  $C_i$  in idle state till next job arrival
- 25 |     |      $C_i$ .SV = MIN VOLTAGE
- 26 |     **end**
- 27 |     **else**
- 28 |     |      $C_i$ .AP =  $C_i$ .AP \*  $C_i$ .SV
- 29 |     |      $C_i$ .SV = nearest higher voltage of ( $C_i$ .AP - J.AET in SF)/( $C_i$ .AP - J.AET in SF +  $C_i$ .NID)
- 30 |     |      $C_i$ .AP =  $C_i$ .AP/ $C_i$ .SV
- 31 |     |     Execute highest priority job at  $C_i$ .SV
- 32 |     **end**
- 33 **end**
- 34 **End Event2**

---

completed, the scheduler invokes **SELECTOR** to find the appropriate voltage and frequency for job execution and executes the highest priority ready job at scaled voltage and frequency. If the AP has not completed, the scheduler executes the highest priority job at previously set voltage and frequency.

**Event 2: On job completion:** On job completion, it updates the execution time of running job, AP and NID. If AP and NID are zero, the scheduler invokes **SELECTOR** algorithm to find the appropriate voltage and frequency for job execution. **SELECTOR** also finds the Procrastinated Idle Duration (PID) as  $C_i.NID$ . If  $C_i.NID$  is more than SDT, it checks the ready queue. If ready queue is empty, it shuts down the core for PID and sets  $C_i.SV$  to maximum voltage. If ready queue is not empty, it executes the highest priority job at maximum voltage and frequency. If  $C_i.NID$  is less than SDT, it checks for jobs in ready queue. If ready queue is empty it keeps the core idle till  $C_i.NID$  and sets  $C_i.SV$  to minimum voltage. If ready queue is not empty, it computes the appropriate voltage and frequency using AP, NID and available voltage/frequency levels.

#### 4.3.2.3 Motivating Example

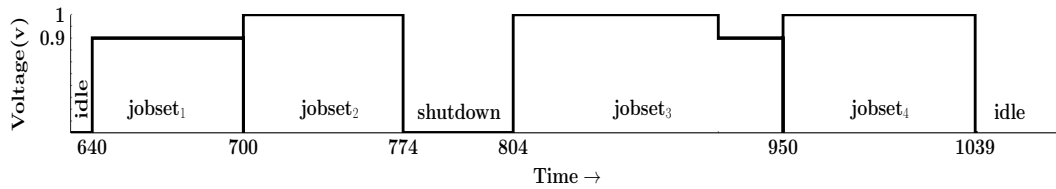
Consider a task set consisting of seven periodic hard real-time tasks with temporal parameters shown in table 4.2. According to MFFBP, two cores are required to execute this task set. Based on period, the tasks are ordered as  $\{ T_0, T_1, T_2, T_4, T_3, T_6 \}$  and  $T_5$ . The task set is divided into two sub task sets  $S_1: \{ T_0, T_1, T_2 \}$  and  $S_2: \{ T_4, T_3, T_6, T_5 \}$ .  $S_1$  and  $S_2$  are allocated to the

Table 4.2: Task set 2

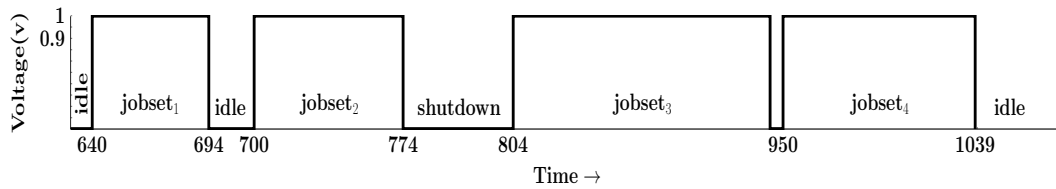
Taskno	Period	WCET
$T_0$	50	20
$T_1$	60	15
$T_2$	80	19
$T_3$	110	15
$T_4$	100	20
$T_5$	140	25
$T_6$	120	20

cores  $C_1$  and  $C_2$  respectively for scheduling. Consider scheduling of jobs of tasks in  $S_1$  with AET same as estimated WCET. At the beginning of the active period, i.e. at  $t=640$ , the SELECTOR algorithm is called. The jobs in  $W$  are  $\{ J_{2,8}, J_{0,13}, J_{1,11} \}$ . The SELECTOR algorithm finds  $C_1.AP$  as 54. It guarantees the initialization of scaled voltage value at the end of each active period. It calls DPS algorithm to compute  $C_1.NID$  at  $t = 640 + 54 = 694$ . In this case, DPS returns 6 as  $C_1.NID$ . Assuming  $C_1.NID$  is more than SDT, it decides to slowdown the core and the  $C_1.SV$  is calculated as  $54/(54+6)=90\%$  of the maximum voltage and frequency. The jobs  $J_{0,13}$ ,  $J_{2,8}$  and  $J_{1,11}$  are executed with reduced voltage and frequency between 640 and 700. At  $t = 700$ , the next active period begins with only one job  $J_{0,14}$  in  $W$ . There is no idle time before  $700 + J_{0,14}.WCET = 720$ . Thus job  $J_{0,14}$  executes with maximum voltage and frequency till 720. At  $t=720$ ,  $C_1.AP = 54$ . At  $t = 720 + 54 = 774$ , DPS computes  $C_1.NID$  as 30 by procrastinating the job that arrives at 780. Assuming  $C_1.NID \geq SDT$ ,  $C_1.SV$  is set to maximum voltage and the tasks are executed with full voltage and frequency from  $t = 700$  till 774. Core  $C_1$  is switched to shutdown state from  $t=774$  till

804. Thus the jobs in active period before core shutdown are executed at maximum voltage and frequency. Using the same method,  $C_1.SV$  is computed as maximum voltage till  $t=920$ , 90% of maximum voltage till  $t=950$  and maximum voltage till  $t=1039$ . Core  $C_1$  remains idle till 1040. The resultant schedule using DPVFS and DPS till time 1040 are shown in Figure 4.16 (a) and (b) respectively.



(a)



(b)

jobset<sub>1</sub>: j<sub>0,13</sub>, j<sub>2,8</sub>, j<sub>1,11</sub>  
 jobset<sub>2</sub>: j<sub>0,14</sub>, j<sub>1,12</sub>, j<sub>2,9</sub>, j<sub>0,15</sub>  
 jobset<sub>3</sub>: j<sub>0,16</sub>, j<sub>2,10</sub>, j<sub>1,13</sub>, j<sub>1,14</sub>, j<sub>0,17</sub>, j<sub>2,11</sub>, j<sub>0,18</sub>, j<sub>1,15</sub>  
 jobset<sub>4</sub>: j<sub>0,19</sub>, j<sub>1,16</sub>, j<sub>2,12</sub>, j<sub>0,20</sub>, j<sub>1,17</sub>

Figure 4.16: (a) Schedule with DPVFS (b) Schedule with DPS

#### 4.3.2.4 Analysis of DPVFS Algorithm

This section describes schedulability, correctness and complexity analysis of DPVFS algorithm.

### Schedulability analysis

**Theorem 4.3.4** *Any periodic task set  $T_i$  with implicit deadlines having less than or equal to 100% utilization is DPVFS schedulable on a core  $C$ .*

**Proof:** DPVFS is a task level dynamic priority scheduling algorithm for individual core with decision points as job arrivals, job completions and core wakeup. For a given task set  $T_i$  on core  $C$ , DPVFS offers a valid schedule if it satisfies the utilization bound given in Equation 4.3 after slowing down the core.

$$U_C = \sum_{i=1}^N U_i / f \leq 1. \quad (4.16)$$

The slowdown decisions in DPVFS make sure it maintains the schedulability bound. Consider figure 4.6. The slowdown decision is made at the beginning of the AP ( $currentAP_{begin}$ ) and on completion of job if it finishes earlier than its WCET thereby leaving slack before the end of AP ( $currentAP_{end}$ ). It is decided to slowdown only if the PI is less than the shutdown threshold. The slack produced by the jobs is utilized to slowdown other jobs till the beginning of next active period ( $nextAP_{begin}$ ). Thus it does not affect the schedulability. The frequency of the core chosen for execution of jobs in  $currentAP$  is

$$f = \sum_{\forall j_i \in J} j_i \cdot AET + \sum_{\forall j_k \in K} j_k \cdot WCET \leq currentAP + PI \quad (4.17)$$

where  $J$  is the set of completed jobs before  $t$  in current active period and  $K$  is the set of incomplete jobs in remaining active period.

Equation 4.17 shows that no higher priority job in  $_{current}AP$  will miss its deadline due to extension of lower priority jobs. Thus the task set remains schedulable after voltage and frequency scaling. Theorem 4.3.1 shows that no job will miss its deadline due to postponement of job execution and the task set remains schedulable with procrastination.

### Correctness analysis

**Theorem 4.3.5** *DPVFS produces a valid and feasible schedule if there exist one.*

**Proof:** For periodic task set with implicit deadlines having hard affinity, any priority driven scheduler is invoked on job arrival and completion. DPVFS also has these decision points but it has to not only select the highest priority job from ready queue but also has to decide the state of the core (slowdown/shutdown). Theorem 4.3.4 shows that a job can be feasibly scheduled even on executing for longer duration without any deadline misses. Theorem 4.3.1 shows that a job can be feasibly scheduled even after postponing its execution without any deadline miss. Thus DPVFS produce a valid and feasible schedule if there exist one in voltage/frequency scale down and procrastination decision.

**Theorem 4.3.6** *DPVFS produces a schedule with equal or lesser energy consumption compared to EDF, DVFS and DPS algorithms.*

**Proof:** Let  $SE_{EDF}$ ,  $SE_{DVFS}$ ,  $SE_{DPS}$  and  $SE_{DPVFS}$  be the static energy consumed and  $DE_{EDF}$ ,  $DE_{DVFS}$ ,  $DE_{DPS}$  and  $DE_{DPVFS}$  be the dynamic energy consumed while scheduling. The trend of static and dynamic energy consumptions of the algorithms is as follows:

$$SE_{DPS} \leq SE_{DPVFS} \leq \{SE_{EDF}, SE_{DVFS}\}$$

$$DE_{DVFS} \leq DE_{DPVFS} \leq DE_{DPS} \leq DE_{EDF}$$

DVFS algorithm takes care of only dynamic energy optimization whereas DPS algorithm takes care of static energy optimization. Since DPVFS follows shutdown whenever possible and slowdown otherwise, it gets advantage of both static and dynamic energy. This results in overall energy reduction compared to other approaches. Thus the overall energy consumption in DPVFS will be lesser than or equal to EDF, DVFS and DPS.

## Complexity analysis

DPVFS is a task level dynamic priority scheduling algorithm with decision points as (i) job(s) arrival, (ii) core wakeup and (iii) job completion. When the core is awake, on job(s) arrival, the job(s) joins the ready queue which is maintained as priority queue in  $N \log N$  time where  $N$  is the number of tasks allocated to the core. Selecting next job to run with same voltage and frequency is a constant ( $C_1$ ) time operation. Thus the run time complexity of job scheduling on job(s) arrival when core is awake is  $\mathcal{O}(N \log N)$ .

If the job(s) arrives when the ready queue is empty, the active period and next idle duration is computed after the job insertion. The active period is computed by considering WCET of all the jobs which arrives before the end of active period. It is seen experimentally that the number of jobs varies from 1 to  $2*N$ . Thus it takes  $\mathcal{O}(2N)$  time to compute active period. The idle duration from the end of active period is computed by considering the WCET of jobs between earliest release time of job after the end of current active period and deadline of the lowest priority job arriving before the nearest deadline ( $D_{next}$ ). It takes  $\mathcal{O}(L \log L)$  for finding and arranging the jobs based on their deadline where  $L$  is the number of jobs arriving between end of active period and  $D_{next}$ . Thus it takes  $\mathcal{O}(L \log L)$  time to compute next idle duration. Thus the run time complexity of job scheduling when ready queue is empty is  $\mathcal{O}(N \log N + L \log L)$ .

Though the transition from shutdown to active is time consuming, for complexity calculation it is considered as constant ( $C_2$ ) time operation. When the core wakes up from shutdown, all the pending jobs joins the priority ready queue in  $N \log N$  times. Selecting next job to run with maximum voltage and frequency is a constant ( $C_1$ ) time operation. Thus the run time complexity of job scheduling on core wake up is  $\mathcal{O}(N \log N)$ .

On job completion, the active and idle periods are updated in constant ( $C_3$ ) time. Then a decision to slowdown or procrastinate is taken in constant time ( $C_4$ ). Accordingly the core is transited to shutdown or idle state and



appropriate voltage level setting in constant ( $C_2$ ) time. If the ready queue is non-empty and the decision is shutting down the core when it becomes idle, the voltage level is set to MAX VOLTAGE. If the decision is to slow down, the appropriate voltage and frequency is computed. Thus like other seminal dynamic priority algorithms, DPVFS also takes constant ( $C_5$ ) time for scheduling a job on job completion. Thus the run time complexity of DPVFS on job completion is  $\mathcal{O}(1)$ . The worst case run time complexity of DPVFS is  $\text{MAX} \{ \mathcal{O}(N \log N + L \log L), \mathcal{O}(N \log N), \mathcal{O}(1) \}$ . In asymptotic notation it is  $\mathcal{O}(N \log N + L \log L)$ .

#### 4.3.2.5 Experimental Evaluation

**Experimental setup:** The experimentation is conducted using the task model described in Chapter 3. The task sets are allocated using MFFBP algorithm described in Section 4.2.1. The framework SMART described in Chapter 3 is used to find the schedule and measure energy parameters using DPVFS, ccEDF and DPS scheduling algorithms.

#### Experimental results:

Figure 4.17 shows the static energy consumption per unit time for different utilizations. DPS consumes the least static energy followed by DPVFS and ccEDF. This is because the static energy consumption is inversely proportional to the shutdown duration. The shutdown duration is more in DPS algorithm compared to DPVFS and ccEDF. This is because DVFS reduces the chance of shutdown as it increases the execution time of jobs. The DPVFS algorithm offers very close performance in comparison with pure procrastination based

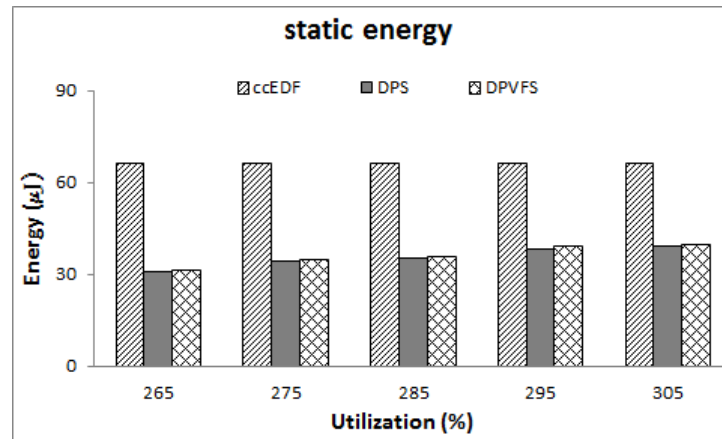


Figure 4.17: Static energy consumption per unit time for different utilizations

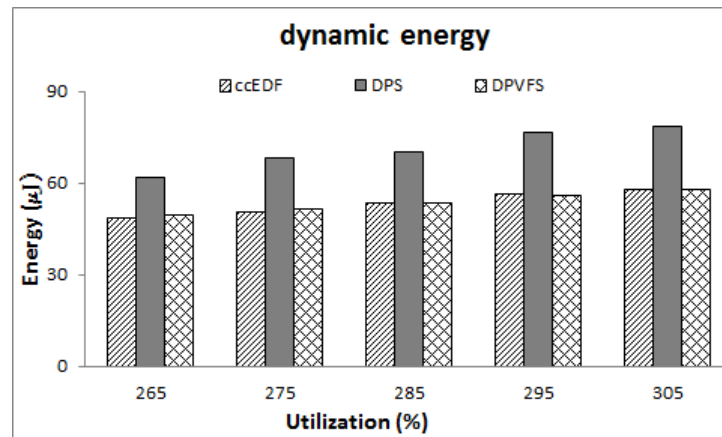


Figure 4.18: Dynamic energy consumption per unit time for different utilizations

algorithms which show its conversion accuracy from idle to shutdown. On an average, DPVFS reduces the static energy by 84.54% over ccEDF and has 1.45% more static energy consumption over DPS.

Figure 4.18 shows the dynamic energy consumption per unit time for different utilizations. Irrespective of the utilizations, ccEDF offers the least dynamic energy consumption followed by DPVFS and DPS algorithms. The DPVFS algorithm offers very close performance in comparison with ccEDF which is because of its aggressive shutdown and slowdown strategies. DPVFS increases dynamic energy by 0.38% over ccEDF and saves 32.7% over DPS.

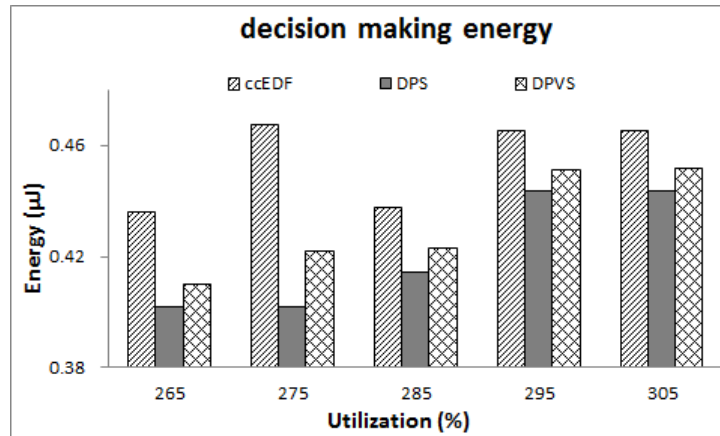


Figure 4.19: Decision making energy per unit time for different utilizations

Figure 4.19 shows the decision making energy per unit time for different utilizations. In procrastination with voltage scaling approach, at the beginning of every active period, current active duration and next available idle duration is computed. Based on this idle duration, active period and scaled voltage is computed. The idle duration computation is also carried out when there

are no jobs in ready queue. This adds into the decision making overhead. Thus DPVFS consumes more energy in decision making compared to DPS. In ccEDF, at every arrival and completion of job, the scaled voltage is computed and thus has more number of decision points compared to DPVFS. DPVFS has decision making overhead of 2.44% over DPS and 5.38% saving over ccEDF respectively.

Figure 4.20 shows the shutdown energy per unit time for different utilizations. The chance of converting inactive period into active period by voltage scaling approach lessens the chance of shutting down the core. DPVFS reduces the shutdown overhead by 4.43% over DPS. Though the decision making and context switching energy are less in procrastination schedulers, it is not significant as compared to static and dynamic energy reductions.

Figure 4.21 shows the idle state energy consumption per unit time for different utilizations. DPVFS consumes the least energy in idle state followed by DPS and ccEDF. This is because DPVFS converts most of the inactive periods into either active or shutdown and leaves very less idle duration whereas ccEDF converts inactive period into active or idle and DPS converts it into shutdown wherever possible. On an average, ccEDF and DPS produces 72% and 56.8% more idle energy respectively than DPVFS.

Figure 4.22 shows the total energy consumption per unit time for different utilizations with shutdown threshold as 500 time units. DPVFS consumes least energy compared to ccEDF and DPS. With the increase in utilization

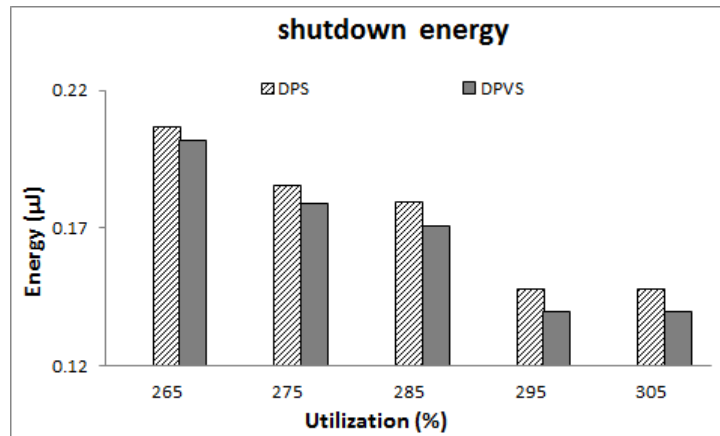


Figure 4.20: Shutdown energy per unit time for different utilizations

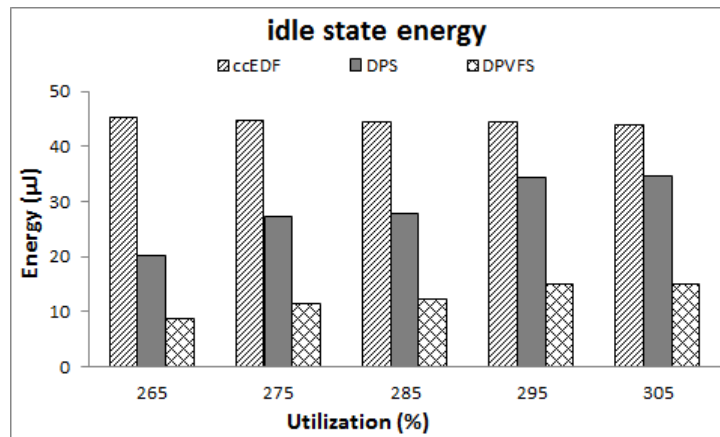


Figure 4.21: Idle state energy per unit time for different utilizations

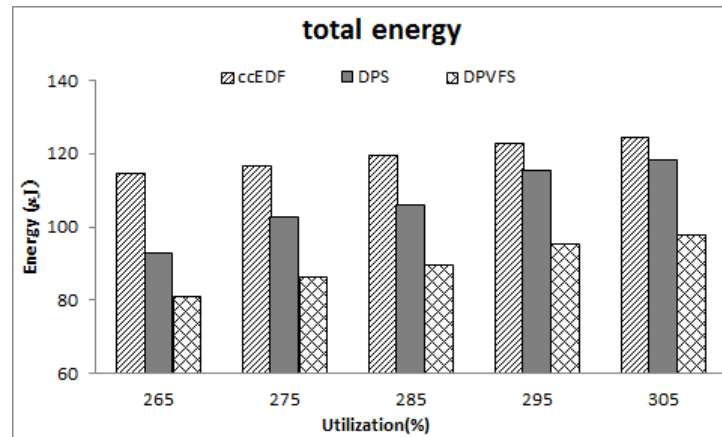


Figure 4.22: Total energy consumption per unit time for different utilizations

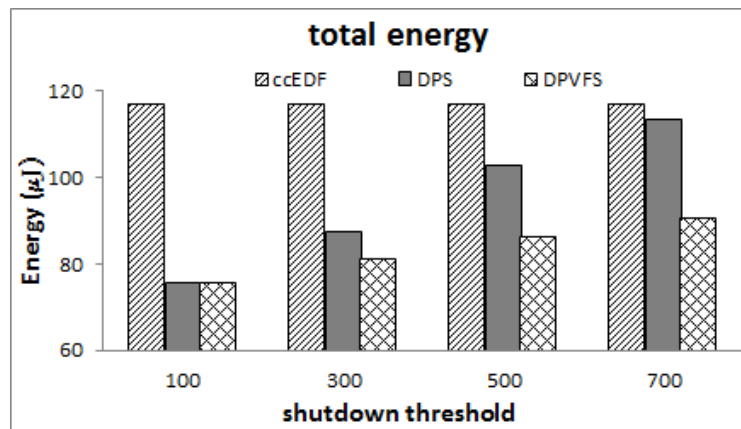


Figure 4.23: Total energy consumption per unit time for different shutdown thresholds

for same shutdown threshold, the chance of shutting down the core reduces since active duration increases. This increases the static energy consumption in DPS algorithm and both static and dynamic energy consumption in ccEDF compared to DPVFS algorithm. Thus DPVFS gives significant saving in total energy compared to DPS algorithm. On an average, DPVFS reduces total energy by 33.2% and 18.8% over ccEDF and DPS respectively.

Figure 4.23 shows the total energy consumption per unit time with different shutdown thresholds for task set having 275% utilization. With the increase in threshold, DPVFS offers significant saving in total energy compared to DPS algorithms. ccEDF has no impact with shutdown threshold as it never allows the core to shutdown. DPVFS saves 54.14% and 0.15% of energy over ccEDF and DPS when shutdown threshold is set to 100. It saves 28.84% and 25.06% of energy over ccEDF and DPS when shutdown threshold is set to 700.

**Conclusion:** Along with the primary constraint of timeliness of real time task, DPVFS scheduler also minimizes overall energy consumption while scheduling by reducing the static and dynamic energy consumption over ccEDF and DPS schedulers with MFFBP task allocation. It is applicable to the MC system supporting various power modes and having discrete levels of voltages and frequencies.

## 4.4 Summary

In this chapter MFFBP - an energy aware task allocation method, DPS - a Dynamic Procrastination Scheduler and DPVFS - a Dynamic Procrastination cum Dynamic Voltage and Frequency Scaling scheduler for MC-HRTS are described, analyzed and evaluated in detail. MFFBP arranges the tasks before allocation such that core with higher period tasks gets more chance for shut down. DPS increases the chances of shutting down by postponing the task execution without violating the timing constraints. Thus saves static and dynamic energy in the schedule. DPS can be applied to the systems that support shutdown and single operational voltage level. DPVFS balances the static and dynamic energy consumption by suggesting the appropriate option of voltage/frequency scaling and procrastination. DPVFS can be applied to the systems that support shutdown and discrete operational voltage levels. Since both the schedulers are dynamic, optimal shutdown duration / optimal voltage scaling can be obtained. The detailed analysis of these algorithms is done based on schedulability, correctness and complexity. DPS and DPVFS produces a valid and feasible schedule on procrastination and voltage/frequency scaling. The run time complexity of DPS algorithm is  $\mathcal{O}(N \log N)$  and that of DPVFS is  $\mathcal{O}(N \log N + L \log L)$ . The schedules produced using DPS and DPVFS with MFFBP task allocation consume less or same energy as various scheduling algorithms- FFEDF\_SD, MFFSTATICPRO, ConvDPS, FFDPS and ccEDF. It is observed that DPS algorithm produces schedule with least energy compared to other algorithms for system with single voltage and frequency and DPVFS algorithm produces



schedule with least energy than other algorithms for system supporting multiple voltage and frequency levels.

# Chapter 5

## Energy Efficient Scheduling in MC-HRTS using Migration

### 5.1 Introduction

This chapter focuses on energy efficient scheduling for MC-HRTS using migration. This chapter elaborates the design of two dynamic schedulers - **OASIS** and **HANDT** for homogeneous MC systems supporting migration. In **OASIS** scheduler, job migration and dynamic procrastination techniques are combined to reduce the static and dynamic energy consumptions. **OASIS** is applicable for systems supporting shutdown. **OASIS** scheduler is extended to **HANDT** for the systems supporting discrete operational voltages and frequencies to further reduce the dynamic energy consumption. In **HANDT** scheduler, DVFS technique is combined with procrastination and migration. In this chapter, the schedulers designed using **OASIS** and **HANDT** are compared with EDF, EDFSD, DPS and ccEDF to evaluate the performance and energy consumption.

## 5.2 Energy Efficient Dynamic Schedulers

Widely used task scheduling schemes in MC system are broadly classified as global, partitioned and semi-partitioned [32] [34]. In global scheme, a global ready queue is used and the tasks are allowed to migrate between cores [35]. In this scheme, affinity of a job to a core is very weak which results in increased cache invalidations. The additional overhead because of cache invalidation may result in deadline misses and/or exceeding energy budget of the system. In partitioned scheme, the non-migratable tasks are statically partitioned and assigned to a specific core for execution [35]. Each core maintains a local ready queue. Due to non-migratable nature, the job remains with the same core on preemption. Due to this, some of the cores may remain idle even when jobs are ready available for execution in other cores. The worst case timing analysis of partitioned approach is much tighter than the global approach as migration related overheads like cache invalidation are eliminated. Semi-partitioned scheme is a hybrid approach in which, initially the tasks are assigned to cores [38] [39]. While scheduling, the jobs are allowed to migrate from one core to other depending on their affinity. This improves the core utilization and balances the workload among cores. The migration in semi-partitioned scheme requires coordination between the cores yielding to high decision making and cache invalidation cost. In real time systems due to timeliness constraint, the overhead of communication between cores need to be avoided. Thus we prefer to use semi-partitioned scheme for OASIS and HANDT schedulers. The migration policy not only helps in improving the performance but also in reducing the energy consumption. OASIS scheduler

is designed to reduce the static energy consumption by reducing the idle duration and increasing the shutdown duration. HANDT scheduler is designed to reduce static and dynamic energy consumption by further reducing the idle durations in the schedules produced by OASIS scheduler.

### 5.2.1 OASIS Scheduler

Motivated with the fact that there is no advantage in completing a job with a hard deadline early, the job execution can be procrastinated. Dynamic Procrastination (DP) is one of the most efficient techniques to optimize the energy consumption during idle period. The distributed idle/shutdown intervals within a core can be combined together with the help of procrastination. DP is effective only when the idle duration is beyond the shutdown threshold. In MC system, the idle intervals are spread across multiple cores which makes it impossible to combine with procrastination. The only possible solution to this problem is to combine procrastination with migration. **OASIS** - an **O**ptimal **s**tatic energy **S**cheduler with **m**Igration and dynamic **p**rocras**t**ination is designed for the same. OASIS aims at optimizing static energy consumption by combining idle intervals spread across cores to longer idle duration. When the source core ( $C_S$ ) becomes idle, OASIS scheduler tries to increase the idle duration by pushing the upcoming jobs to other core(s) where it has sufficient slack for execution. For the job which cannot be migrated, OASIS scheduler tries to procrastinate upcoming job execution to further increase the idle duration. If the push and procrastinate action benefits in producing the idle duration more than the shutdown threshold, the scheduler shuts down the core. This shutdown duration helps

in effective static energy saving. If push migration and procrastination do not benefit in shutdown duration, the core has to remain idle till the next job arrival. The core consume energy during this unused idle duration. OASIS scheduler converts these idle durations into active by pulling and executing the jobs from other cores. This helps in increasing the utilization of active core(s). The conversion of idle period into active helps in effective energy saving. The accumulated idle time in other core(s) can be converted into shutdown using procrastination. The decision of migrating the jobs to/from other cores is done using **Push - Procrastinate - Pull (P<sup>3</sup>)** policy. P<sup>3</sup> policy is designed to dynamically recommend the choice of pushing or pulling of jobs based on the available idle duration and the shutdown threshold.

#### 5.2.1.1 OASIS algorithm

Algorithms [5], [6], [7], [8] and [9] shows OASIS, P<sup>3</sup> POLICY, FINDPUSHABLEJOBS, PROCRASTINATE and FINDPULLABLEJOBS respectively.

The data structures used in the algorithms are:

- Lcores[] - list of cores
- C<sub>i</sub>.MQ - Migrated queue of core C<sub>i</sub> that holds the migrated jobs
- C<sub>i</sub>.Ljobs[] - list of jobs in core C<sub>i</sub> in non-decreasing order of release time
- C<sub>i</sub>.LMJ[] - list of migratable jobs and the corresponding target core C<sub>T</sub> to/from which jobs are migrated

The abbreviations used in the algorithms are:

- C<sub>i</sub>.PID - Procrastinated Idle Duration in core C<sub>i</sub>

- $C_i$ .SIT - Scheduler Invocation Time in core  $C_i$
- SDT - ShutDown Threshold
- $C_S$  - Source core
- $C_T$  - Target core

In Push migration, the core from which the job is migrated is represented as source core  $C_S$  and the core to which the job is pushed is represented as target core  $C_T$ . In Pull migration, the core to which the job is migrated is represented as source core  $C_S$  and the core from which the job is pulled is represented as target core  $C_T$ .

**Algorithm OASIS:** OASIS algorithm takes task sets and migrated queue of all cores as input and takes scheduling decision. On arrival of a job  $J$ , it updates the execution time of running job, inserts  $J$  in priority queue and selects the highest priority job from the queue for execution. Similarly on core wake up and on job completion, OASIS scheduler selects the highest priority job from the ready queue for execution. On job completion, if the ready queue is empty, OASIS follows  $P^3$  policy. Based on the state of core, OASIS keeps the core either in idle state or in shutdown state till the next scheduler invocation time.

**$P^3$  Policy:**  $P^3$  calls Algorithm [7]: `FINDPUSHABLEJOBS` to get the list of migratable jobs. It calls Algorithm [8]: `PROCRASTINATE` to get the PID. Based on this duration,  $P^3$  decides whether to push these migratable jobs or to pull the jobs from other cores. If significant length of idle duration

---

**Algorithm 5: OASIS**


---

**Input:**  $\forall_{i=1 \text{ to } m} C_i.MQ[], C_S.RQ[] T_{C_i}$

**Output:** feasible schedule if  $\forall U_{C_i} \leq 1$

```

1 Event1: On Arrival of job J
2 Update Running Job's (R) remaining execution time
3 Insert job J in priority ready queue  $C_S.RQ[]$ 
4 Select the highest priority job from  $C_S.RQ[]$ 
5 End Event1
6 Event2: On core  $C_S$  wake up
7 Select the highest priority job from  $C_S.RQ[]$ 
8 End Event2
9 Event3: On Completion of job J
10 if ( $C_S.RQ[]$  is nonempty) then
11 |   Select the highest priority job from  $C_S.RQ[]$ 
12 else if ( $C_S.RQ[]$  is empty) then
13 |    $C_S.state = \text{Call } P^3(C_S, t)$ 
14 |   if ( $C_S.state == \text{Shutdown}$ ) then
15 |     |   Keep core  $C_S$  in shutdown till  $C_S.SIT$ 
16 |   else if ( $C_S.state == \text{Idle}$ ) then
17 |     |   Keep core  $C_S$  in Idle state till  $C_S.SIT$ 
18 end
19 End Event3

```

---

is available, the migratable jobs are pushed to corresponding target cores, the SIT of target cores is updated and the state of the source core  $C_S$  is set to shutdown. Otherwise  $P^3$  calls Algorithm [9]: FINDPULLABLEJOBS to get the list of migratable jobs from other cores. It pulls the migratable jobs from corresponding target cores to migrated queue of  $C_S$ , updates the SIT of target cores and sets the state of core as idle.  $P^3$  returns the state of source core and SIT.

---

**Algorithm 6: PUSH-PROCRASTINATE-PULL ( $P^3$ ) POLICY**


---

**Input:**  $\forall_{i=1 \text{ to } m} C_i.MQ[], T_{C_i}$

**Output:**  $C_S.state$

```

1  $C_S.LMJ[] = \text{FINDPUSHABLEJOBS}(C_S, t)$ 
2  $C_S.PID = \text{PROCRASTINATE}(C_S, C_S.LMJ[], t)$ 
3 if ( $C_S.PID \geq SDT$ ) then
4    $C_T.MQ[] \leftarrow C_S.LMJ[]$  and update  $C_T.SIT$ 
5    $C_S.state = \text{shutdown}$ 
6    $C_S.SIT = t + C_S.PID$ 
7 end
8 else
9    $\langle C_S.LMJ[], C_S.SIT \rangle = \text{FINDPULLABLEJOBS}(C_S, C_S.SIT)$ 
10   $C_S.MQ[] \leftarrow C_S.LMJ[]$  and update  $C_T.SIT$ 
11   $C_S.state = \text{idle}$ 
12 end
13 return  $C_S.state$  and  $C_S.SIT$ 

```

---



**Algorithm FINDPUSHABLEJOBS:** FINDPUSHABLEJOBS algorithm takes task sets allocated and migrated queue of all cores. It finds the utilization of each core and sorts the list of cores in non-increasing order of their utilization. For each job, it finds the appropriate core having sufficient slack. If the job cannot be migrated due to insufficient slack, the scheduler stops to find the pushable job. The slack is computed by considering the portion of execution times of running job, ready jobs, migrated jobs and future jobs. It returns the list of migratable jobs with corresponding target cores.

---

**Algorithm 7:** FINDPUSHABLEJOBS
 

---

**Input:**  $C_S.Ljobs[]$ ,  $\forall_{i=1 \text{ to } m} C_i.MQ[]$ ,  $T_{C_i}$

**Output:**  $C_S.LMJ[]$

```

1 Find utilization of each core
2 Sort cores in non-increasing order of utilization and store in  $Lcores[]$ 
3 foreach job  $J_i \in C_S.Ljobs[]$  do till migratable
4   foreach core  $C_T \in Lcores[]$  do
5     if ( $C_T.state$  is shutdown) then
6       |  $r = C_T.CWT$ 
7     else
8       |  $r = J_i.r$ 
9       | compute  $C_T.slack$  between  $r$  &  $J_i.d$ 
10      if ( $C_T.slack \geq J_i.WCET$ ) then
11        |  $C_S.LMJ[] \leftarrow J_i$  and break
12      end
13 end
14 return  $C_S.LMJ[]$ 

```

---

---

**Algorithm 8:** PROCRASTINATE

---

**Input:**  $C_S.Ljobs[]$ ,  $C_S.LMJ[]$  and time  $t$  when  $C_S$  became idle**Output:**  $C_S.PID$  - Procrastinated idle duration of  $C_S$ 

```

1  $ND_1 \leftarrow$  absolute deadline of highest priority job after  $t$ 
2 if ( $ND_1 - t - J.WCET < SDT$ ) then
3   | SIT = next job arrival time
4 end
5 else
6   |  $ND_2 \leftarrow$  deadline of lowest priority job arriving before  $ND_1$ 
7   |  $L[] \leftarrow$  list of jobs releasing between  $t$  and  $ND_2$  and  $\notin C_S.LMJ[]$ ,
   |   arranged in non-increasing order of their absolute deadlines
8   | foreach job  $J_i \in L[]$  do
9     | if ( $J_i.d > ND_2$ ) then
10    |   SIT -= (( $ND_2 - J_i.r$ )*( $J_i.WCET / J_i.P$ )* $U_p$ )
11    |   else
12    |     | SIT -=  $J_i.WCET$ 
13    |     | if ( $J_i \neq lastjob \ \&\& \ SIT > J_{i+1}.d$ ) then
14    |     |   | SIT =  $J_{i+1}.d$ 
15    |     | end
16    |   end
17 end
18  $C_S.PID = SIT - t$ 
19 return  $C_S.PID$ 

```

---

**Algorithm PROCRASTINATE:** PROCRASTINATE algorithm takes list of migratable jobs and the task sets allocated to each core and computes the slack for highest priority job. If the slack is not sufficient to shutdown the core, it sets next job arrival time as the SIT. Otherwise it tries to procrastinate the execution of upcoming jobs. It computes PID by considering the jobs arriving between current time  $t$  and  $ND_2$ , where  $ND_2$  is the deadline of the lowest priority job arriving before the nearest deadline  $ND_1$ . The jobs in the list of pushable jobs are not considered while computing PID. All the jobs having deadline before  $ND_2$  are considered with their WCET. For the jobs having deadlines after  $ND_2$ , only a portion of their executions before  $ND_2$  are considered. PROCRASTINATE algorithm returns PID.

**Algorithm FINDPULLABLEJOBS:** FINDPULLABLEJOBS algorithm takes SIT of source core  $C_S$ , list of jobs released before SIT in sorted order of their release times, task sets allocated and migrated queue of all cores. It finds the utilization of each core and sorts the cores in non-increasing order of their utilization. It finds the list of jobs from other cores having sufficient slack in  $C_S$ . With these migratable jobs, it finds the next job arrival as the SIT for  $C_S$ . It returns the list of migratable jobs with corresponding target cores and SIT of  $C_S$ .

### 5.2.1.2 Motivating Example

Consider a task set consisting of six periodic hard real-time tasks with temporal parameters shown in table 5.1. The tasks  $T_0$ ,  $T_1$ ,  $T_2$  and  $T_4$  are allocated to core  $C_1$  and tasks  $T_3$  and  $T_5$  are allocated to core  $C_2$  for scheduling. Consider

**Algorithm 9: FINDPULLABLEJOBS**

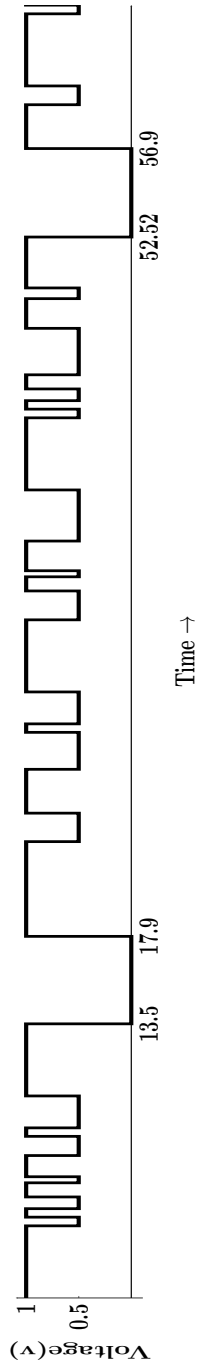

---

**Input:**  $C_S.SIT$ ,  $\forall_{i=1 \text{ to } m} C_i.Ljobs[]$ ,  $C_i.MQ[]$ ,  $T_{C_i}$   
**Output:**  $C_S.LMJ[]$ ,  $C_S.SIT$

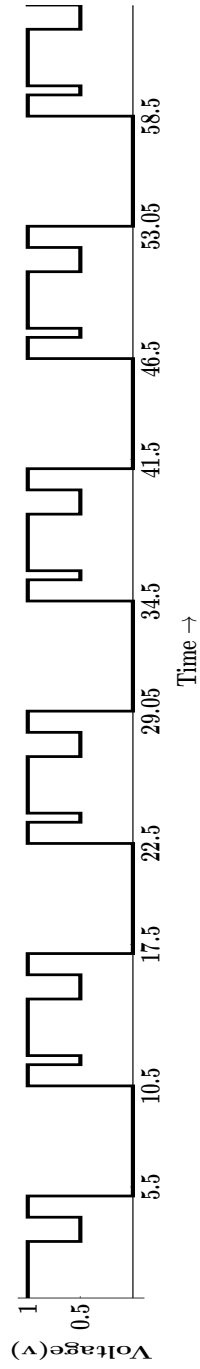
- 1 Find utilization of each core
- 2 Sort cores in non-decreasing order of utilization and store in  $Lcores[]$
- 3 **foreach** core  $C_T \in Lcores[]$  **do**
- 4     **foreach** job  $J_i \in C_T.Ljobs[] : J_i.r < C_S.SIT$  **do**
- 5         compute  $C_S.slack$  between  $J_i.r$  &  $J_i.d$
- 6         **if** ( $C_S.slack \geq J_i.WCET$ ) **then**
- 7              $C_S.LMJ[] \leftarrow J_i$
- 8         **end**
- 9 **end**
- 10  $C_S.SIT =$  next job release time
- 11 **return**  $C_S.LMJ[]$  and  $C_S.SIT$

---

scheduling of jobs with AET as 70% of estimated WCET. At  $t=28$ , core  $C_2$  becomes idle. The next job  $J_{3,1}$  arrives at 40. Assuming the idle duration ( $40 - 28 = 12$ ) is less than SDT (40), OASIS invokes FINDPUSHABLEJOBS. FINDPUSHABLEJOBS finds core  $C_1$  with sufficient slack for next arriving job  $J_{3,1}$ . The next arriving job  $J_{3,2}$  cannot be migrated due to insufficient slack. The FINDPUSHABLEJOBS invokes PROCRASTINATE. PROCRASTINATE finds  $ND_1$  and  $ND_2 = 120$  and procrastinates job  $J_{3,2}$  till 105. It computes PID as  $105 - 28 = 77$  units. It returns job  $J_{3,1}$  as migratable job on core  $C_1$  and PID as 77 units. As PID is more than SDT, OASIS scheduler decides core  $C_2$  to shutdown till 105. At  $t=115.5$ , core  $C_2$  becomes idle. OASIS invokes FINDPUSHABLEJOBS to find slack for next arriving job  $J_{0,9}$  in core  $C_1$ . Core  $C_1$  do not have sufficient slack. The FINDPUSHABLEJOBS returns empty list

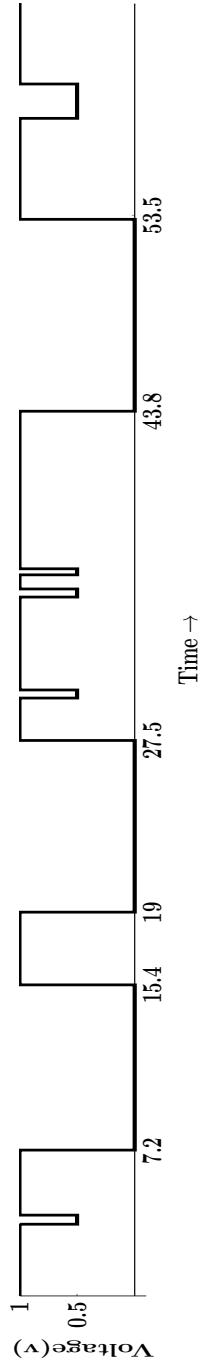


(a) core 0 in ConvDPS schedule

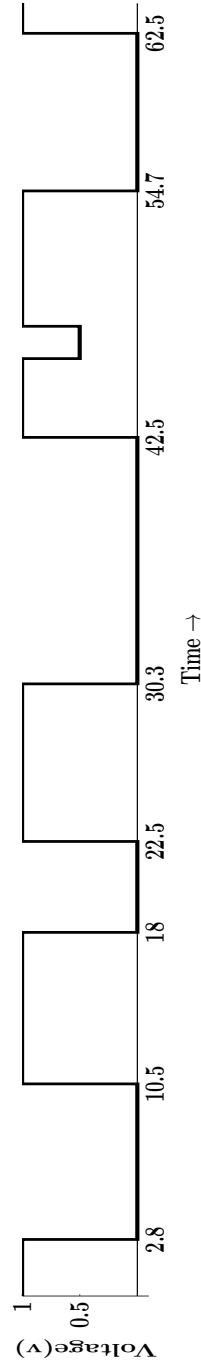


(b) core 1 in ConvDPS schedule

Figure 5.1: Core State transitions



(c) core 0 in OASIS schedule



(d) core 1 in OASIS schedule

Fig 5.1 Core State transitions

Table 5.1: Task set 3

Taskno	Period	WCET
$T_0$	40	6
$T_1$	50	10
$T_2$	60	15
$T_3$	40	15
$T_4$	100	20
$T_5$	120	25

of migratable jobs and PID of 4.5 units. As PID is less than SDT, OASIS invokes `FINDPULLABLEJOBS`. `FINDPULLABLEJOBS` computes slack in  $C_2$  for next arriving jobs in ready queue, migrated queue and future jobs of  $C_1$ . It finds sufficient slack for jobs  $J_{0,3}$ ,  $J_{1,3}$  and  $J_{4,2}$ . No migrated jobs are to be checked for pulling since there have been already some jobs pushed by  $C_2$ . `FINDPULLABLEJOBS` returns these three jobs as migratable jobs on core  $C_2$  and its SIT as 120. OASIS computes SIT of  $C_1$  as 118.2. The resultant core state transitions using Conventional Dynamic Procrastination Scheduler (ConvDPS) [26] and OASIS for one hyper-period are shown in Figures 5.1 (a), (b), (c) and (d). The state of core is represented as shutdown, idle and active when voltage is 0v, 0.5v and 1v respectively as shown on y axis. It can be observed that the shutdown duration in schedule with OASIS has increased and idle duration has reduced than the schedule with ConvDPS.

### 5.2.1.3 Analysis of OASIS Algorithm

#### Schedulability analysis

**Theorem 5.2.1** *For a periodic task set with implicit deadlines having soft affinity, OASIS produces a valid and feasible schedule if there exist a feasible*

*schedule by any dynamic priority driven scheduling approach.*

**Proof:** Since each core in OASIS follows static task allocation and dynamic priority scheduling, a periodic task set with implicit deadlines can be feasibly scheduled on the core to which it is allocated such that it follows eq.(4.3). The task set remains schedulable on migration and procrastination if each core follows eq.(5.1).

$$U_C = \sum_{\forall J_{ik} \in T_i} U_{ik} + \sum_{\forall J_{pk} \in LMJ[]} U_{pk} \leq 1. \quad (5.1)$$

The necessary and sufficient condition for migrating any job  $J_S$  that belongs to source core  $C_S$  is to have sufficient slack in the core to which it is allocated. Thus

$$C_T.slack \geq J.WCET \quad (5.2)$$

where

$$C_T.slack = J_S.d - J_S.r + \sum_{\forall J_T \in Cjobs[]} J_T.WCET \quad (5.3)$$

$$J_T.WCET = \frac{(\min(J_S.d, J_T.d) - \max(J_S.r, J_T.r, C_T.wakeuptime))}{J_T.p} * RET \quad (5.4)$$

$Cjobs[] = (C_T.running \text{ job if any}) + (\text{jobs in } C_T.MQ, C_T.RQ) + (C_T.future \text{ jobs releasing before } J.d)$

and  $RET = \text{remaining execution time of } J_T$



Push and Pull migration in OASIS do not affect schedulability of the task set if each target core follows eq.(4.3) and (5.2) after push and/or pull migration. The push, procrastinate and pull decisions in OASIS scheduler make sure it maintains the schedulability bound after following  $P^3$  policy. On procrastination, all the tasks in next active period will be executed with full frequency.

$$j_k.WCET + Z_k + X + Y \leq PI + \text{next}AP \quad (5.5)$$

where

$$X = \sum_{\forall j_i \in J_i: i \neq k \& j_i.r, j_i.d \leq D} j_i.WCET \quad (5.6)$$

$$Y = \sum_{\forall j_l \in J_l: l \neq k \& j_l.r \leq D, j_l.d \geq D} D - j_l.r \quad (5.7)$$

$D$  is the absolute deadline of job having release time  $j_j.r < j_k.d$  and deadline  $j_j.d > j_k.d$ ,

$J_i$  is the set of jobs in next active period ( $\text{next}AP$ ) released before  $D$  and having deadline before  $D$ ,

$J_l$  is the set of jobs in  $\text{next}AP$  released before  $D$  and having deadline after  $D$ .

Equation 5.5 shows that no job will miss its deadline due to postponement of job execution and the task set remains schedulable with procrastination. The pull migration decision is made if  $PID$  is not sufficient for shutdown. The tasks are migrated to the source core only if it follows equation 5.1. Thus the task set remains schedulable after applying  $P^3$  policy.

## Correctness analysis

**Theorem 5.2.2** *The overall shutdown duration produced by the OASIS scheduler is greater than or equal to the one produced by DPS and EDF scheduling algorithms.*

**Proof:** Let  $S_{DPS}$  and  $S_{OASIS}$  be the schedules produced by DPS and OASIS scheduling algorithms. Let  $SD_{DPS}$  and  $SD_{OASIS}$  be the total shutdown duration in  $S_{DPS}$  and  $S_{OASIS}$  respectively. DPS algorithm procrastinates the execution of future jobs and merges the intermediate idle intervals within same core which results into longer shutdown duration. The migration of jobs from other cores and procrastination of upcoming jobs by OASIS increases the procrastinated idle duration to get better overall shutdown duration. Thus

$$SD_{OASIS} \geq SD_{DPS} \quad (5.8)$$

Eq.(4.7) and (5.9)  $\implies$

$$SD_{OASIS} \geq SD_{DPS} \geq SD_{EDF} \quad (5.9)$$

**Theorem 5.2.3** *The schedule produced by the OASIS scheduler offers better energy saving compared to DPS and EDF scheduling algorithms.*

**Proof:** Let  $ID_{DPS}$  and  $ID_{OASIS}$  be the total idle duration in  $S_{DPS}$  and  $S_{OASIS}$  respectively. The objective of OASIS to keep the core in shutdown state instead of idle or execute the jobs from other cores in idle period is to produce overall less idle duration compared DPS. As active duration is constant in both the approaches,

$$SD_{DPS} + ID_{DPS} = SD_{OASIS} + ID_{OASIS} \quad (5.10)$$

Eq.(5.10)  $\implies$

$$total\_shut\_down\_duration \propto \frac{1}{total\_idle\_duration} \quad (5.11)$$

Eqs.(5.8) and (5.11)  $\implies$

$$ID_{OASIS} \leq ID_{DPS} \quad (5.12)$$

Let  $SE_{DPS}$ ,  $SE_{OASIS}$  be the static energy consumed and  $DE_{DPS}$ ,  $DE_{OASIS}$  be the dynamic energy consumed while scheduling  $S_{DPS}$  and  $S_{OASIS}$  respectively.

Eq.(5.8) and (5.12)  $\implies$

$$SE_{OASIS} \leq SE_{DPS} \quad (5.13)$$

and

$$DE_{OASIS} \leq DE_{DPS} \quad (5.14)$$

Let  $NSD_{DPS}$ ,  $NSD_{OASIS}$  be the number of shutdown intervals and  $SDE_{DPS}$ ,  $SDE_{OASIS}$  be the shutdown and wakeup energy in  $S_{DPS}$  and  $S_{OASIS}$  respectively. The prolonged shutdown duration in OASIS due to migration gives less number of shut down intervals compared DPS. Thus Eq.(5.8)  $\implies$

$$NSD_{OASIS} \leq NSD_{DPS} \quad (5.15)$$

Eq.(5.15)  $\implies$

$$SDE_{OASIS} \leq SDE_{DPS} \quad (5.16)$$

Let  $ENERGY_{DPS}$ ,  $ENERGY_{EDF}$  and  $ENERGY_{OASIS}$  be the overall energy while scheduling  $S_{DPS}$ ,  $S_{EDF}$  and  $S_{OASIS}$  respectively. Equations (4.11),

(5.13), (4.12), (5.14), (4.14) and (5.16)  $\implies$

$$ENERGY_{OASIS} \leq ENERGY_{DPS} \leq ENERGY_{EDF} \quad (5.17)$$

Thus the overall energy consumption with OASIS scheduling algorithm is less in comparison with DPS and EDF approaches.

**Theorem 5.2.4** *In high utilization scenarios, OASIS offers better system utilization over all static allocation methods.*

**Proof:** Due to semi-static partitioning in task allocation, (i) OASIS gives better system utilization compared to any static partitioning approach in MC system and (ii) In high utilization situated, OASIS is adaptive to utilize the idle time whenever not feasible to convert into shutdown by migrating jobs from other cores. Thus OASIS offers better energy savings without compromising on utilization compared to other approaches in MC system.

### Complexity analysis

OASIS is a task level dynamic priority scheduling algorithm with events as (i) job(s) arrival (ii) core wakeup and (iii) job completion. The run time complexity of OASIS is  $\text{Max}\{ A_1, A_2, A_3 \}$  where  $A_1, A_2$  and  $A_3$  are the run time complexities of events i, ii and iii respectively.

**Run time complexity of Event (i):**  $A_1$  is with reference to lines 2 to 4 of OASIS algorithm,  $A_1$  is the time required for updation, maintaining priority ready queue and selection which is  $N \log N + \text{constant time}$ .

**Run time complexity of Event (ii):**  $A_2$  is with reference to line 7 of OASIS algorithm,  $A_2$  is the time taken for selection of job which is a constant time operation(C).

**Run time complexity of Event (iii):**  $A_3$  is with reference to lines 10 to 17 of OASIS algorithm,  $A_3 = \text{Max}\{(\text{complexity of lines 10,11}),(\text{complexity of lines 12 to 17})\} = \text{Max}\{(\text{constant time}), (A_4 + \text{constant time})\} = A_4$ , where  $A_4$  is the run time complexity of  $P^3$  algorithm.

**Run time complexity of  $A_4$ :** With reference to  $P^3$  algorithm,  
 Line 1: run time complexity of **FINDPUSHABLEJOBS** as  $A_5$ .  
 Line 2: run time complexity of **PROCRASTINATE** algorithm as  $A_6$ .  
 Lines 3 to 12:  $A_7$  as  $\text{Max}\{(\text{lines 3 to 6}),(\text{lines 7 to 11})\}$ ,  
 $A_7 = \text{Max}\{(\text{constanttime}), (A_8 + \text{constanttime})\} = A_8$  where  
 $A_8$  is the run time complexity of **FINDPULLABLEJOBS** algorithm,  
 So,  $A_4 = A_5 + A_6 + A_8$

**Run time complexity of  $A_5$ :** With reference to **FINDPUSHABLEJOBS** algorithm,

Line 1: Finding utilization of P cores, each with N tasks - PN

Line 2: Sorting based on utilization -  $P \log P$

Lines 5 to 9: The slack for job j is computed by considering the remaining execution time of running job, WCET of ready jobs and WCET of

migrated/future jobs that arrives before the deadline of job  $j$ . It is seen experimentally that the number of jobs considered for slack calculation varies from 1 to  $2N$ . Thus run time complexity for slack calculation is  $2N + \text{constant } C$ .

Lines 10 and 11: On finding sufficient slack, the job is added to the list of migratable jobs in constant time.

Lines 4 to 12: The work has restricted to check the slack for  $2*N$  migratable jobs.

Lines 3 to 13: The slack is checked in  $P$  cores.

$$\text{So, } A_5 = PN + P \log P + P * (2N + C) + C = PN + PN^2$$

**Run time complexity of  $A_6$ :** Run time complexity  $A_6$ : With reference to **PROCRASTINATE** algorithm,

$$A_6 = \text{line 1} + \text{Max}\{(\text{lines 2 to 4}), (\text{lines 5 to 17})\} + \text{line 18},$$

Line 1: Finding absolute earliest deadline among  $L$  jobs -  $L$

Lines 2 to 4: Checking for sufficient idle duration and finding scheduler invocation time in constant time  $C$ .

$$\text{Lines 5 to 17: } L + \text{Computation of procrastinated idle duration using } L \text{ jobs} \\ + L = 2L + L \log L$$

Line 18: constant time for computation of possible idle duration

$$\text{So, } A_6 = L + \text{Max}\{(C), (2L + L \log L)\} + C = 3L + L \log L + C = L \log L$$

**Run time complexity of  $A_8$ :** With reference to **FINDPULLABLEJOBS** algorithm,

Line 1: Finding utilization of  $P$  cores, each with  $N$  tasks -  $PN$

Line 2: Sorting based on utilization -  $P \log P$

Line 5: Slack computation in  $2N + C$  time

Lines 6 and 7: On finding sufficient slack, the job is added to the list of migratable jobs in constant time

Lines 4 to 8: The work has restricted to check the slack for  $2*N$  migratable jobs

Lines 3 to 9: The slack is checked in  $P$  cores

$$\text{So, } A_8 = NP + P \log P + 2N * P * (2N + C) + C = 2NP + N^2P$$

$$\text{So, } A_3 = A_4 = A_5 + A_6 + A_8 = (NP + N^2P) + (L \log L) + (2NP + N^2P) = N^2P + L \log L$$

Thus the run time complexity of OASIS =  $Max\{A_1, A_2, A_3\} = Max\{N \log N, C, PN^2 + L \log L\} = PN^2 + L \log L$ . In asymptotic notation it is  $\mathcal{O}(PN^2 + L \log L)$ .

#### 5.2.1.4 Experimental Evaluation

**Experimental setup:** The experimentation is conducted using the task model described in Chapter 3. The framework SMART described in Chapter 3 is used to find schedule and measure energy parameters like inactive, static, dynamic and total energy consumptions by schedules produced using EDF, EDF with shutdown (EDFSD), Conventional DPS (ConvDPS), DPS, and OASIS schedulers.

**Experimental results:** Let  $S_{EDF}$ ,  $S_{EDFSD}$ ,  $S_{ConvDPS}$  and  $S_{OASIS}$  be the schedules using EDF without shutdown, EDFSD, ConvDPS and OASIS

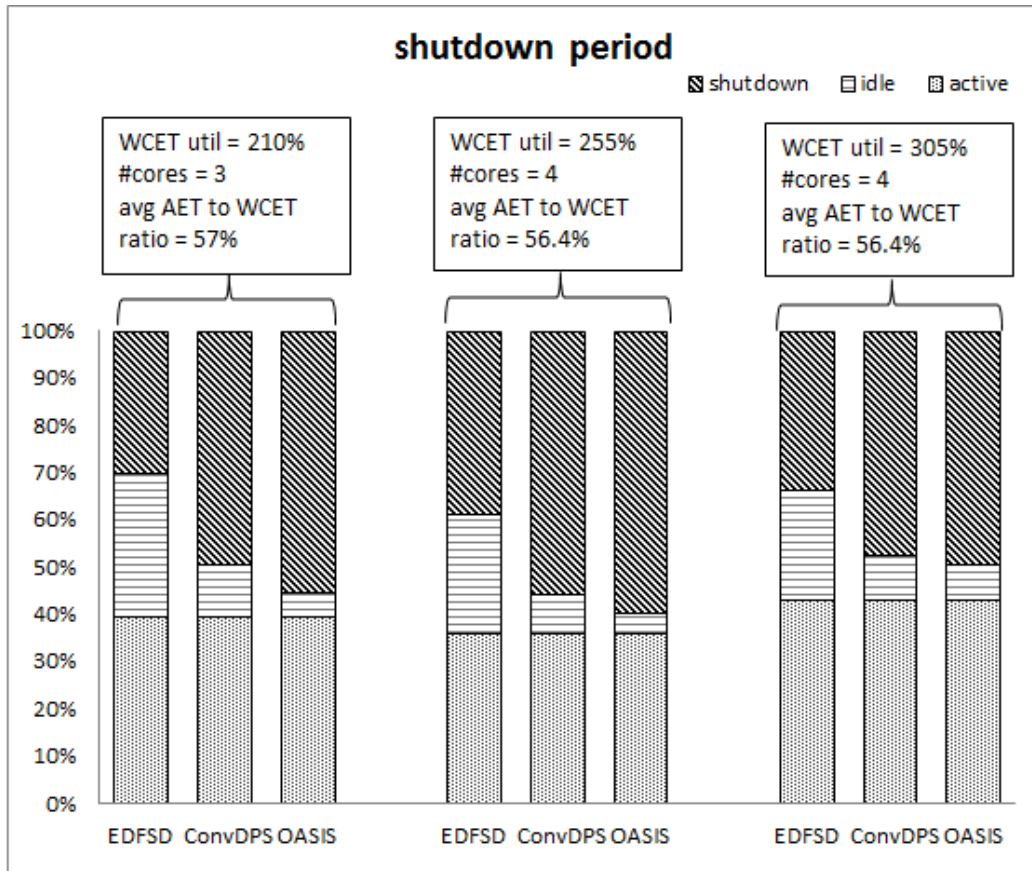


Figure 5.2: Percentage of active, idle and shutdown period for varying utilizations

schedulers. Figure 5.2 shows percentage of active, idle and shutdown period in  $S_{EDFSD}$ ,  $S_{ConvDPS}$  and  $S_{OASIS}$  for utilizations 210%, 255% and 305% for 3, 4 and 4 cores respectively. Migration in OASIS increases the procrastination duration and converts idle duration into shutdown duration. On an average, OASIS produces 38.24% and 6.8% shutdown period over EDFSD and ConvDPS schedules respectively.



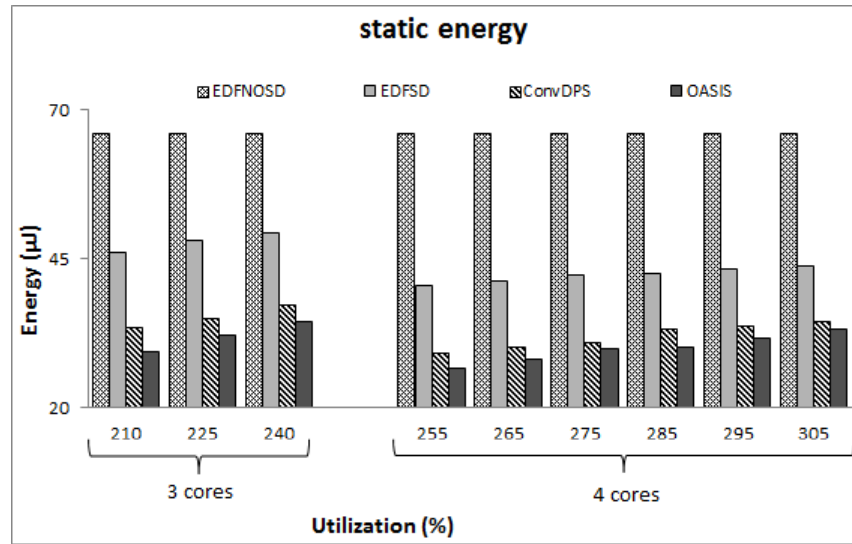


Figure 5.3: Static energy consumption per unit time for varying utilization

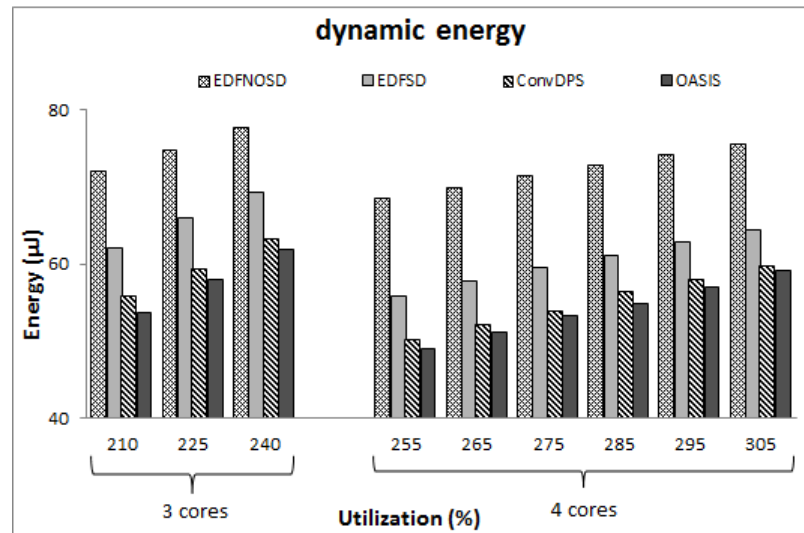


Figure 5.4: Dynamic energy consumption per unit time for varying utilization

Figure 5.3 and Figure 5.4 shows the total static energy consumption and total dynamic energy consumption per unit time respectively by  $S_{EDF}$ ,  $S_{EDFSD}$ ,

$S_{ConvDPS}$  and  $S_{OASIS}$  for varying utilizations. Irrespective of utilizations,  $S_{EDF}$  consumes same static energy.  $S_{OASIS}$  consumes the least static and dynamic energy followed by  $S_{ConvDPS}$ ,  $S_{EDFSD}$  and  $S_{EDF}$ . In OASIS, most of the inactive idle durations are converted into shutdown or active due to procrastination and migration. This increases the overall shut down duration and reduces the overall idle duration. As static and dynamic energy is directly proportional to shutdown and idle duration, overall static and dynamic energy in OASIS reduces. On an average, OASIS reduces the static energy by 116.5%, 44.2% and 8% over EDF, EDFSD and ConvDPS algorithms respectively. On an average, OASIS reduces the dynamic energy by 32.22%, 12.22% and 2.18% over EDF, EDFSD and ConvDPS algorithms respectively.

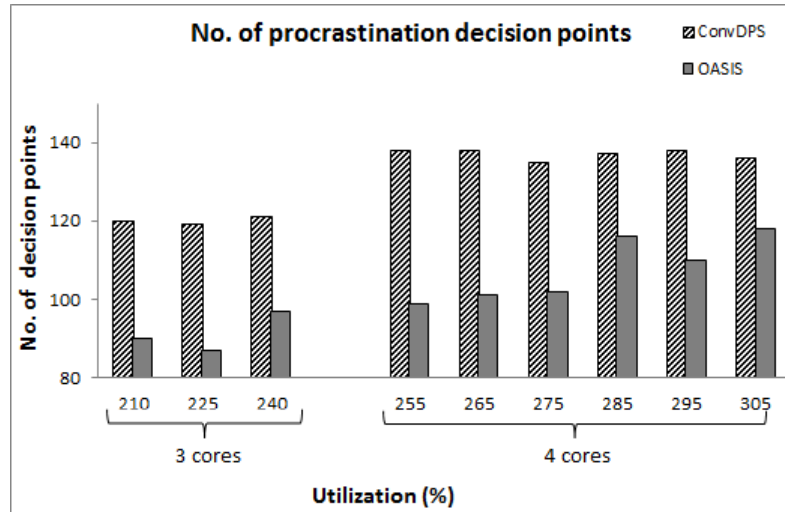


Figure 5.5: Number of procrastination decision making points for varying utilizations

Figure 5.5 shows the energy consumption for procrastination decisions in

OASIS and ConvDPS for varying utilizations. In ConvDPS and OASIS, the procrastination decision is taken when there are no jobs in ready queue. This decision is based on the idle duration computed by procrastinating the future jobs and the shutdown threshold. In OASIS, the idle duration computation also includes procrastination duration due to push migration. Whenever push migration is not beneficial, the idle duration is computed based on pulled jobs. OASIS also includes re-computation of scheduler invocation time of cores to-which/from-where the jobs are migrated. This adds into the procrastination decision making overhead in OASIS and ConvDPS. Migration in OASIS merges inactive states and reduces the overall procrastination decision points. Thus OASIS consumes less energy in procrastination decision making compared to ConvDPS. On an average, OASIS reduces procrastination decision energy by 29.11% over ConvDPS algorithm.

Figure 5.6 shows the energy consumption of OASIS, ConvDPS and EDFSD due to shutdown and wakeup activities. The chance of converting idle period into shutdown by procrastination and migration approach increases the chance of shutting down the core. Procrastination with migration approach of OASIS reduces overall shutdown states over ConvDPS approach. Thus the shutdown overhead in OASIS is 28.2% more than EDFSD but 1.6% less than ConvDPS.

Figure 5.7 shows the total energy consumption per unit time by EDF, EDFSD, ConvDPS and OASIS during inactive duration for varying utilizations which includes static, dynamic, procrastination decision making, shutdown and

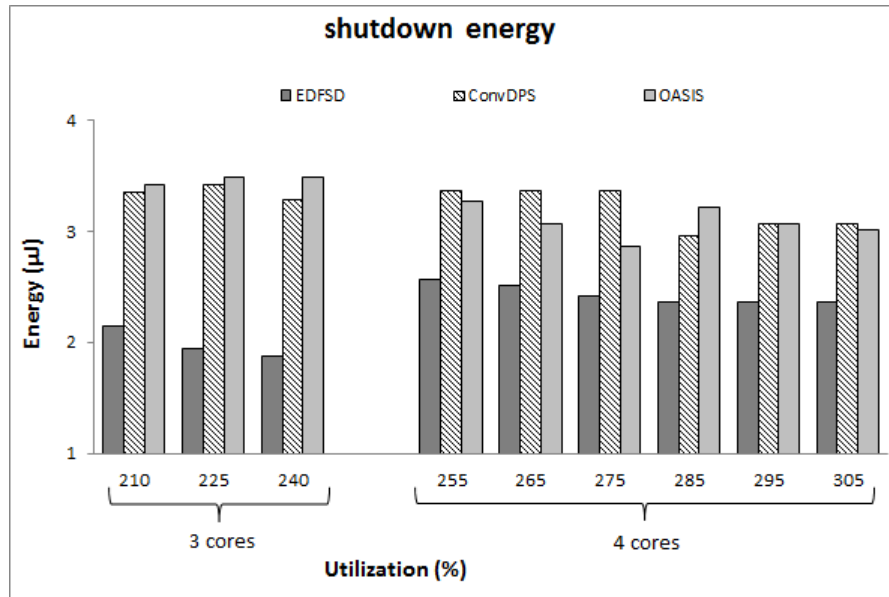


Figure 5.6: Shutdown overhead per unit time for varying utilizations

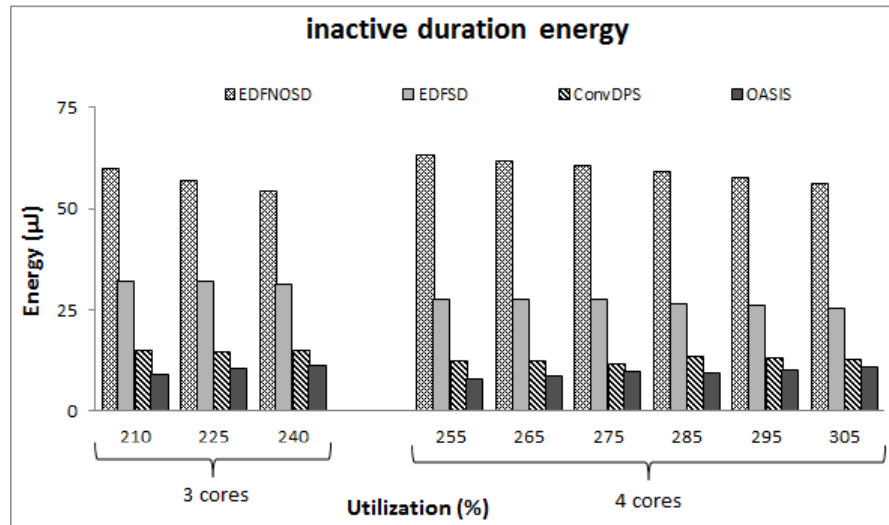


Figure 5.7: Total inactive duration energy consumption per unit time for varying utilizations

wakeup energy. OASIS schedule has less idle duration and more shutdown duration compared to all other schedules. Thus OASIS schedule consumes the least energy in inactive duration followed by ConvDPS, EDFSD and EDF schedules. On an average, OASIS schedule consumes 517%, 196.4% and 40.04% less energy in inactive period than EDF, EDFSD and ConvDPS schedules respectively.

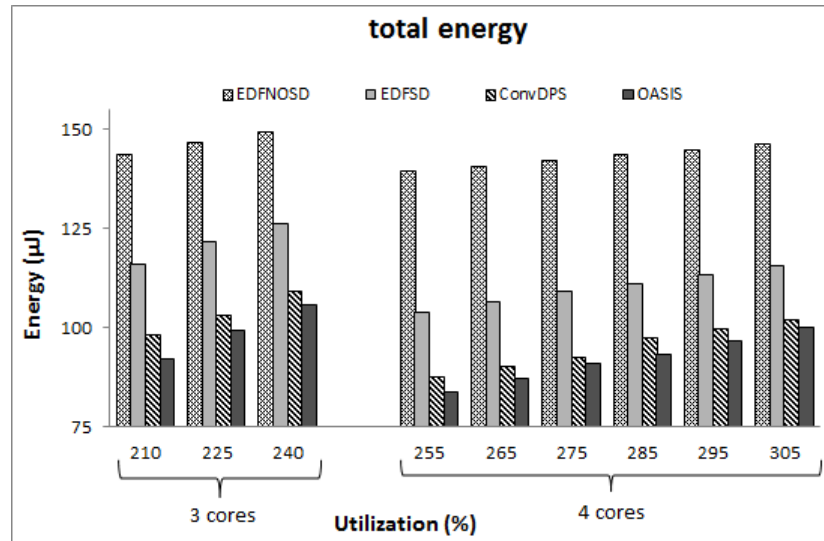


Figure 5.8: Total energy consumption per unit time for varying utilizations

Figure 5.8 shows the total energy consumption per unit for varying utilizations. Due to increase in overall shutdown duration, the static and dynamic energy consumptions are reduced in OASIS compared to EDF and ConvDPS approaches. On an average, OASIS reduces overall energy consumption by 53.44%, 20.7% and 3.88% over EDF, EDFSD and ConvDPS respectively.

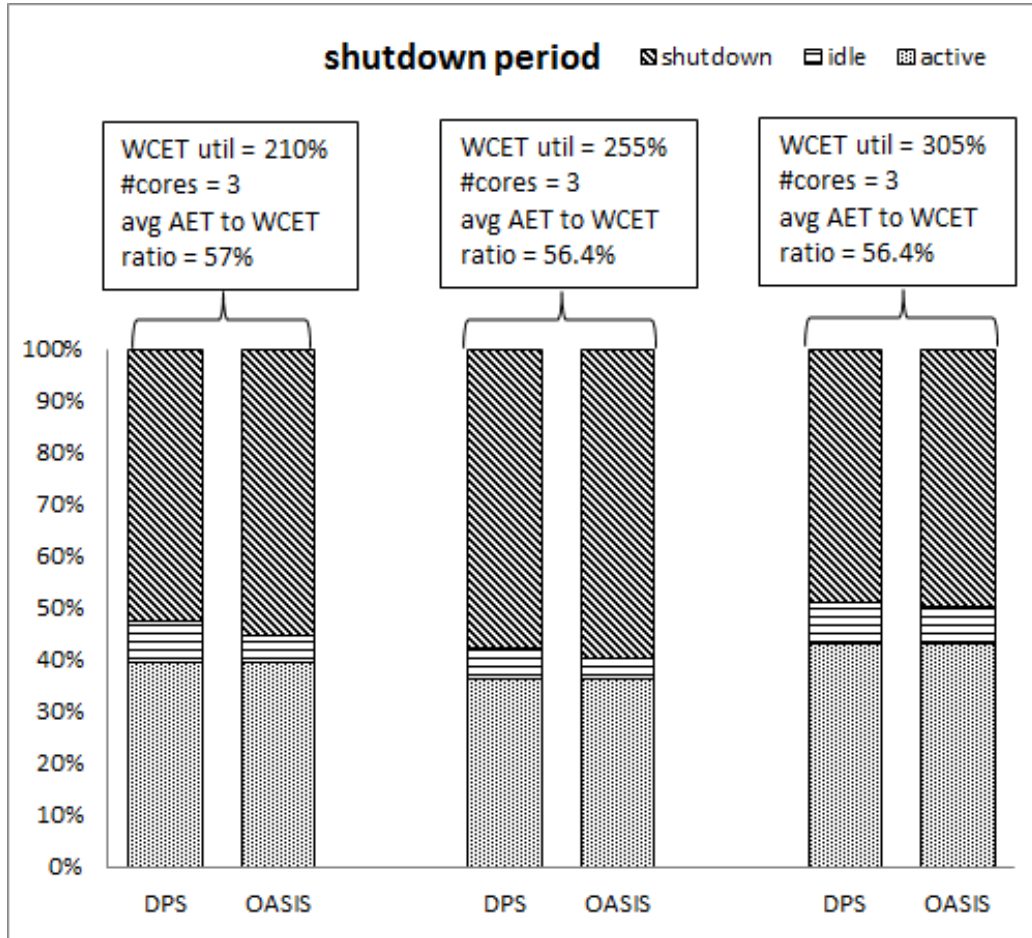


Figure 5.9: Percentage of shutdown period over idle period for varying utilizations

Let  $S_{DPS}$  be the schedule produced using proposed DPS scheduler (described in Chapter 4). Figure 5.9 shows percentage of shutdown period over idle period in  $S_{DPS}$  and  $S_{OASIS}$  for varying utilizations. Migration in OASIS increases the procrastination duration and converts idle duration into shutdown duration. As most of the idle duration in  $S_{OASIS}$  is either converted into shutdown or active, it has more shutdown duration compared to idle duration in complete schedule. On an average, OASIS produces 2.22% shutdown period DPS schedule.

Figure 5.10 and Figure 5.11 show the total static energy consumption and total dynamic energy consumption per unit time respectively by  $S_{DPS}$  and  $S_{OASIS}$  for varying utilizations. On an average, OASIS reduces the static and dynamic energy by 2.55% and 0.7% respectively over DPS algorithm.

Figure 5.12 shows the energy consumption for procrastination decisions in  $S_{OASIS}$  and  $S_{DPS}$  for varying utilizations. On an average, OASIS reduces procrastination decision energy by 12.33% over DPS algorithm.

Figure 5.13 shows the energy consumption in OASIS and DPS due to shutdown and wakeup activities. The shutdown overhead in OASIS is 1.11% less than DPS schedules.

Figure 5.14 shows the total energy consumption per unit time by DPS and OASIS during inactive duration for varying utilizations. On an average, OASIS schedule consumes 13.37% less energy in inactive period than DPS

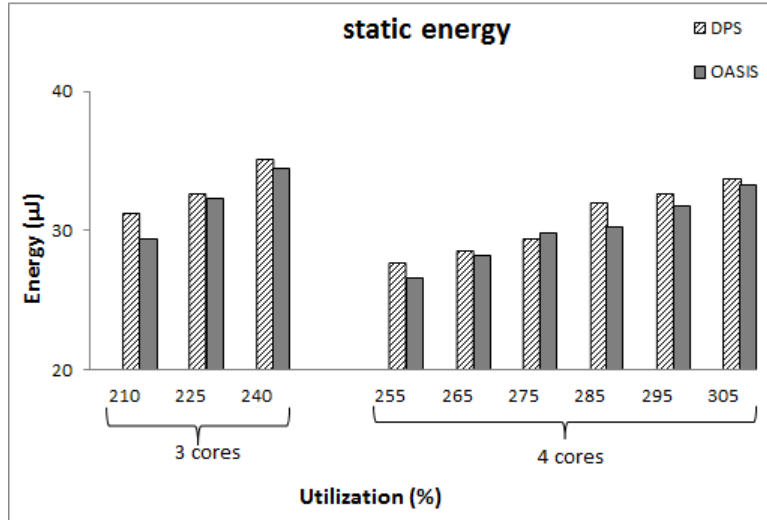


Figure 5.10: Static energy consumption per unit time for varying utilizations

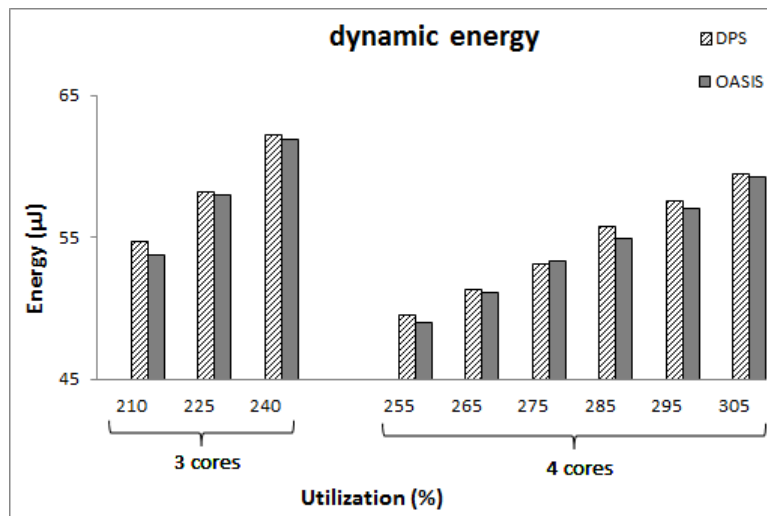


Figure 5.11: Dynamic energy consumption per unit time for varying utilizations



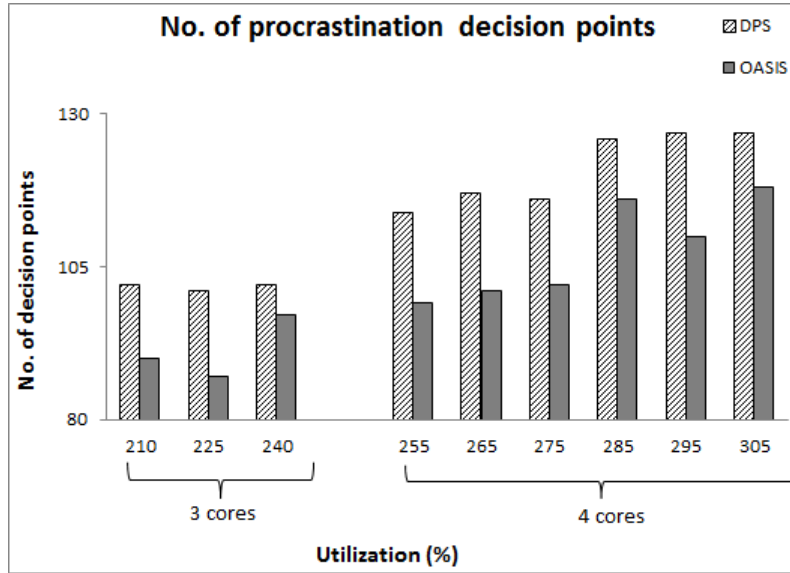


Figure 5.12: Number of procrastination decision making points for varying utilizations

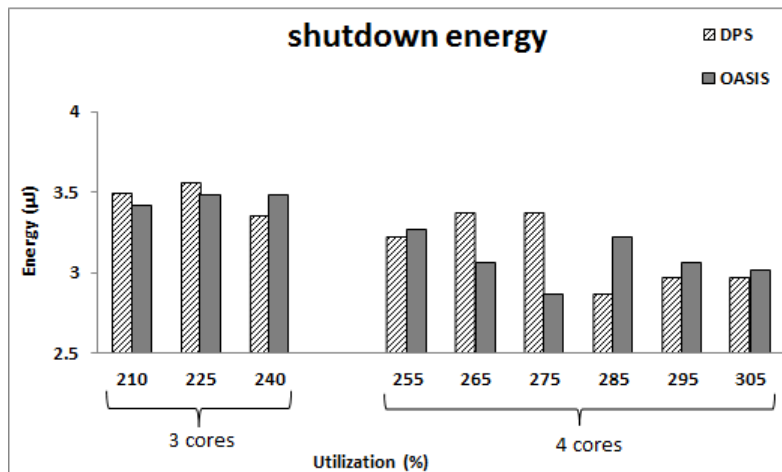


Figure 5.13: Shutdown overhead per unit time for varying utilizations

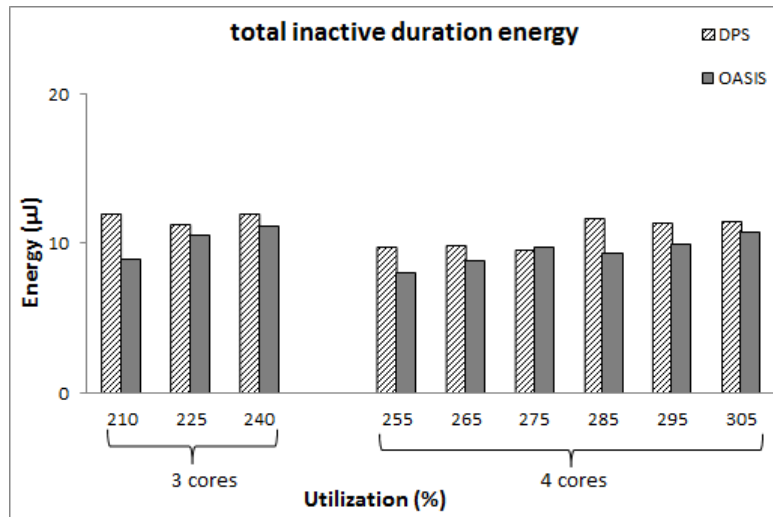


Figure 5.14: Total inactive duration energy consumption per unit time for varying utilizations

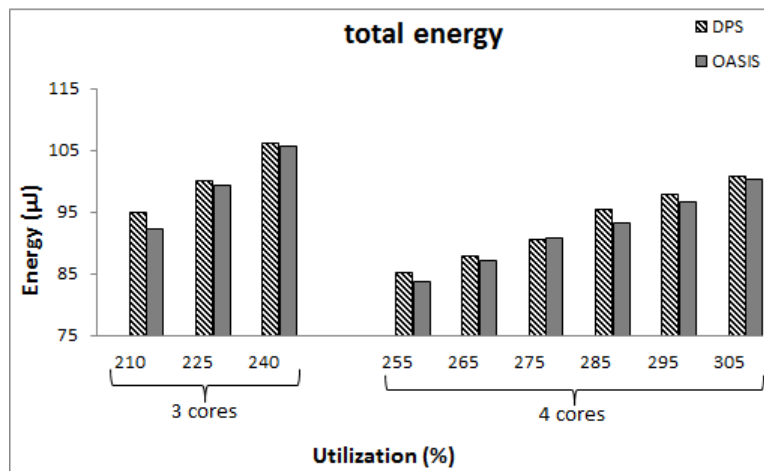


Figure 5.15: Total energy consumption per unit time for varying utilizations

schedules.

Figure 5.15 shows the total energy consumption per unit for varying utilizations. On an average, OASIS reduces overall energy consumption by 1.2% over DPS schedule. It can be observed that OASIS algorithm gives closer values to proposed DPS algorithm but better than Conventional DPS algorithm.

**Conclusion:** Along with the primary constraint of timeliness on real time tasks, OASIS scheduler also minimizes overall energy consumption by reducing the static and dynamic energy consumption over EDF, EDF with shutdown, ConvDPS and DPS algorithms. It is applicable to the MC systems supporting various operating modes and migration.

### 5.2.2 HANDT scheduler

OASIS improves static energy saving using dynamic procrastination technique and migration. This effectively executes the tasks at maximum voltage and frequency. Motivated with the same concept that there is no advantage in completing a job with a hard deadline early, the run time slack can be utilized by slowing down the job executions at reduced voltage till it is safe to do. This method of scaling down the supply voltage and operating frequency (DVFS) effectively reduces dynamic energy consumption. DVFS can be done in systems that support multiple voltage levels and frequencies. The scaled voltage/frequency is computed with WCET. In practice small idle intervals may get formed because of run time slack and non-availability

of continuous voltage and frequency levels. In MC system, these unused idle durations can be utilized by applying job migration techniques. This may effectively produce longer idle duration in some cores for shutdown and thus reduces overall energy consumption further. The concept of task migration is particularly used in distributed operating systems and parallel computing domain for fault tolerance and load balancing. Controlled job migration can also be used effectively in MC system for optimizing energy consumption. **HANDT (Hare AND Tortoise)** scheduler is designed for homogeneous MC-HRTS supporting discrete voltage and frequency levels and multiple power modes like active, idle and shutdown. This work also considers semi-partitioned task allocation with soft affinity.

HANDT scheduler adopts the mechanisms followed by hare and tortoise of Aesop's fables to complete all the jobs without deadline miss. If the system has prolonged inactive time, HANDT scheduler adopt hare policy of finishing all the existing jobs at the earliest to go for a long nap, i.e. to shut down for a longer duration to save both static and dynamic energy. When the system has only short bursts of inactive periods, it follows tortoise policy of reducing the speed, thus energy consumption. HANDT scheduler use DP and shutdown as techniques to implement hare policy. It uses DVFS as a technique to implement tortoise policy. HANDT uses job migration along with DP and DVFS techniques for optimizing energy consumption. This is achieved by migrating the jobs to other cores and shutting the cores down for maximum possible duration. The unproductive idle time is utilized by slowing down the core or by migrating and executing the jobs from other

cores so as to shutdown the other cores if possible. Migration with DP and DVFS helps in increasing the core utilization and shutdown duration which results in saving static and dynamic energy consumption. DVFS incurs energy overhead due to voltage transitions. Though these techniques work well independently, combining them to get optimized overall energy consumption without negatively affecting the performance of an application in MC system is still a challenge. The energy-performance ratio of a MC system can be improved on selected cores by combining various techniques like DVFS and DP along with migration. This introduces challenges like meeting deadlines, dynamic job migration, optimal shutdown duration, optimal voltage/frequency scaling, optimal static and dynamic energy consumption, energy consumption due to preemptions and cache impact etc.

At the start of active period, HANDT scheduler uses **RECOMMENDER** algorithm to decide whether to execute jobs in active period at maximum voltage/frequency or scaled down voltage/frequency. **RECOMMENDER** algorithm computes the idle duration at the end of active period by considering the WCET of jobs in active period. If idle duration is sufficient for shutting down the core, **RECOMMENDER** sets the scaled voltage to maximum. If idle duration is not sufficient for shutting down the core, it finds the jobs that can be pushed to other cores and procrastinate the non migratable job such that the push and procrastinate action can increase the idle duration. If the push migration and procrastination does not give sufficient time for shutting down the core, **RECOMMENDER** finds the jobs from the end of active period that can be pushed to other cores and helps in increasing the idle duration. If there is

sufficient increase in duration to shutdown the core, RECOMMENDER sets the scaled voltage to maximum. If the idle duration by push and procrastination is not sufficient to shutdown the core, RECOMMENDER finds the appropriate voltage and frequency for job executions and scales down the voltage/frequency accordingly. The scheduler executes the jobs in active period at the voltage set by RECOMMENDER. The increased shutdown duration helps in static energy saving. As most of the jobs completes before the estimated WCET, the ready to run jobs can be executed at reduced voltage and frequency. Utilizing the idle period effectively reduces the dynamic energy consumption. RECOMMENDER algorithm dynamically suggests HANDT scheduler whether to execute the jobs in active period at maximum voltage or slow down the core, without violating the timing constraints. When the core becomes idle, HANDT scheduler decides the state of the core using P<sup>3</sup> policy described in section (5.2.1.1).

### 5.2.2.1 HANDT algorithm

Algorithms [10], [11] and [12] shows HANDT, RECOMMENDER and FINDPUSHABLEJOBSREVERSE algorithms respectively.

The data structures used in the algorithms are:

- Lcores[] - list of cores
- C<sub>i</sub>.MQ - Migrated queue of core C<sub>i</sub> that holds the migrated jobs
- C<sub>i</sub>.Ljobs[] - list of jobs in core C<sub>i</sub> in non-decreasing order of release time
- C<sub>i</sub>.LMJ[] - list of migratable jobs and the corresponding C<sub>T</sub> to/from which jobs are migrated

The abbreviations used in the algorithms are:

- $C_i$ .AP - Current Active Period in core
- stolenAP - Stolen Active Period
- EAP - End of Active Period
- $C_i$ .NID - Next Idle Duration in core  $C_i$  at EAP
- NSAP - Next Start of Active Period
- $C_i$ .PID - Procrastinated Idle Duration in core  $C_i$
- $C_i$ .SIT - Scheduler Invocation Time in core  $C_i$
- SDT - ShutDown Threshold

**Algorithm HANDT:** HANDT algorithm takes task set and migrated queue of all cores as input and generates the valid schedule with voltage scaling or procrastination. It schedules the jobs on following four events:

**Event 1: Beginning of active period:** At the beginning of active period, the scheduler calls RECOMMENDER algorithm to find scaled voltage  $C_S$ .SV. HANDT executes the highest priority ready job at  $C_S$ .SV.

**Event 2: Job arrival:** On job arrival, it updates the execution time of running job, active period (AP), next idle duration (NID) and inserts new job in priority queue. If AP is completed, the scheduler invokes SELECTOR algorithm to find the appropriate voltage and frequency for job execution and executes the highest priority ready job at scaled voltage and frequency. The SELECTOR algorithm is described in Section 4.3.2. If the AP has not

**Algorithm 10: HANDT**


---

**Input:**  $\forall_{i=1 \text{ to } m} C_i.MQ[], T_{C_i}$   
**Output:** Schedule with voltage scaling or procrastination

- 1 **Event1: At the start of active period**
- 2  $C_S.SV = \text{Call RECOMMENDER}(C_S)$
- 3 Execute highest priority job J at  $C_S.SV$
- 4 **End Event1**
- 5 **Event2: On Arrival of job J**
- 6 Update R.WCET,  $C_S.AP$  and  $C_S.NID$  with R.AET in SF
- 7 Insert job J in priority ready queue  $C_S.RQ[]$
- 8 **if** ( $C_S.AP == 0$ ) **then**
- 9     |  $\text{SELECTOR}(C_S)$
- 10    | Execute job J at  $C_S.SV$ ;
- 11 **end**
- 12 **else**
- 13    | Execute highest priority job at  $C_S.SV$
- 14 **End Event2**
- 15 **Event3: On core wake up**
- 16 Execute the highest priority job from  $C_S.RQ[]$  at maximum voltage and frequency
- 17 **End Event3**
- 18 **Event4: On Completion of job J**
- 19 **if** ( $C_S.RQ$  is empty) **then**
- 20    |  $C_S.state = \text{Call } P^3(C_S, t)$
- 21    | **if** ( $C_S.state == \text{Shutdown}$ ) **then**
- 22      | Keep core  $C_S$  in shutdown till  $C_S.SIT$
- 23    | **else if** ( $C_S.state == \text{Idle}$ ) **then**
- 24      | Keep core  $C_S$  in Idle state till  $C_S.SIT$
- 25 **end**
- 26 **else**
- 27    | Update R.WCET,  $C_S.AP$  &  $C_S.NID$  with R.AET in SF
- 28    | **if** ( $(C_S.AP == 0) \ \&\& \ (C_S.NID == 0)$ ) **then**
- 29      |  $\text{SELECTOR}(C_S)$
- 30    |  $C_S.AP = C_S.AP * C_S.SV$
- 31    |  $C_S.SV = \text{nearest higher voltage of } (C_S.AP - J.AET \text{ in SF}) / (C_S.AP - J.AET \text{ in SF} + C_S.NID)$
- 32    |  $C_S.AP = C_S.AP / C_S.SV$
- 33    | Execute highest priority job at  $C_S.SV$
- 34 **end**
- 35 **End Event4**

---



completed, the scheduler executes the highest priority job at previously set voltage and frequency.

**Event 3: Core wake up:** On core wake up, HANDT scheduler executes the highest priority ready job at maximum voltage and frequency.

**Event 4: Job completion:** On job completion, if the ready queue is empty, HANDT follows  $P^3$  policy.  $P^3$  policy is described in Section (5.2.1.1).  $P^3$  returns the state of the source core and SIT to HANDT. Based on the state of core, HANDT keeps the core either in idle or shutdown state till  $C_S.SIT$ . If the ready queue is non empty on job completion, the HANDT scheduler updates the AP and NID. If both are zero, the scheduler invokes SELECTOR algorithm to find the appropriate voltage and frequency for job execution and executes the highest priority job at scaled voltage and frequency.

**Algorithm RECOMMENDER:** The RECOMMENDER algorithm computes AP and NID. If NID is more than SDT, it sets  $C_S.SV$  to maximum. Otherwise it calls the Algorithm [7]: FINDPUSHABLEJOBS. The FINDPUSHABLEJOBS returns the list of migratable jobs with corresponding target cores. RECOMMENDER calls the Algorithm [8]: PROCRASTINATE to compute Procrastinated Idle Duration (PID). RECOMMENDER algorithm computes Next Start of Active Period (NSAP). Based on PID, RECOMMENDER decides whether to push these migratable jobs or needs to find some more jobs to be pushed. If significant length of idle duration is available, the migratable

**Algorithm 11: RECOMMENDER**


---

**Input:**  $\forall_{i=1 \text{ to } m} C_i.MQ[], T_{C_i}$   
**Output:**  $C_S.SV$

- 1 Compute  $C_S.AP$  and  $C_S.NID$
- 2 **if** ( $C_S.NID \geq SDT$ ) **then**
- 3 |  $C_S.SV = \text{MAX VOLTAGE}$
- 4 **end**
- 5 **else**
- 6 | Compute end of active period  $EAP = t + C_S.AP$
- 7 |  $C_S.LMJ[] = \text{FINDPUSHABLEJOBS}(C_S, t)$
- 8 |  $C_S.PID = \text{PROCRASTINATE}(C_S, C_S.LMJ[], t)$
- 9 | Compute next start of active period  $NSAP = EAP + C_S.PID$
- 10 **if** ( $C_S.PID \geq SDT$ ) **then**
- 11 |  $C_T.MQ[] \leftarrow C_S.LMJ[]$  and update  $C_T.SIT$
- 12 |  $C_S.NID = C_S.PID$
- 13 **end**
- 14 **else**
- 15 |  $ReqdExtraID = SDT - C_i.PID$
- 16 |  $stolenAP = \text{FINDPUSHABLEJOBSREVERSE}(C_S, \text{REQDEXTRAID})$
- 17 | **if** ( $stolenAP \geq ReqdExtraID$ ) **then**
- 18 | |  $C_T.MQ[] \leftarrow C_S.LMJ[]$
- 19 | |  $C_T.MQ[] \leftarrow C_S.LMJReverse[]$
- 20 | | Recompute  $C_S.AP$
- 21 | | Compute  $C_S.NID = NSAP - (t + C_S.AP)$
- 22 | **end**
- 23 **end**
- 24 **if** ( $C_S.NID \geq SDT$ ) **then**
- 25 |  $C_S.SV = \text{MAX VOLTAGE}$
- 26 **end**
- 27 **else**
- 28 |  $C_S.SV = \text{nearest higher voltage of } (C_S.AP) / (C_S.AP + C_S.NID)$
- 29 **end**
- 30 **end**
- 31 **return**  $C_S.SV$

---

jobs are pushed to corresponding target cores. The scheduler invocation time of target cores is updated and sets NID as computed PID. Otherwise RECOMMENDER calls the Algorithm [12]: FINDPUSHABLEJOBSREVERSE to compute the extra idle duration (ReqdExtraID) required for shutting down the core. It returns the  $C_S.LMJReverse[]$  and stolen active period (stolenAP). If stolenAP is more than the ReqdExtraID, the jobs from  $C_S.LMJ[]$  and  $C_S.LMJReverse[]$  are migrated to the corresponding target cores and the  $C_S.AP$  and  $C_S.NID$  is updated. If  $C_S.NID$  is more than SDT, RECOMMENDER sets  $C_S.SV$  to maximum. Otherwise it sets  $C_S.SV$  to available appropriate voltage. It returns scaled voltage  $C_S.SV$ .

**Algorithm FINDPUSHABLEJOBS:** The FINDPUSHABLEJOBS algorithm finds the utilization of each core and sorts the list of cores in non-increasing order of their utilization. For each job, it finds the appropriate core having sufficient slack. If the job cannot be migrated due to insufficient slack, the scheduler stops to find the pushable job. The slack is computed by considering the portion of execution times of running job, ready jobs, migrated jobs and future jobs. It returns the list of migratable jobs with corresponding target cores.

**Algorithm PROCRASTINATE:** The PROCRASTINATE algorithm takes the list of migratable jobs and task sets allocated to each core. It computes the slack for highest priority job. If the slack is not sufficient to shutdown the core, it sets the scheduler to invoke on next job arrival. Otherwise it tries to procrastinate the execution of upcoming jobs. PROCRASTINATE algorithm

computes the Procrastinated Idle Duration (PID) by considering the jobs arriving between current time  $t$  and  $ND_2$ .  $ND_2$  is the deadline of the lowest priority job arriving before the nearest deadline  $ND_1$ . The jobs in the list of pushable jobs are not considered while computing the procrastination duration. All the jobs having deadline before  $ND_2$  are considered with their WCET. For the jobs whose deadlines are after  $ND_2$ , only a portion of their executions before  $ND_2$  are considered. PROCRASTINATE algorithm returns PID.

**Algorithm FINDPUSHABLEJOBSREVERSE:**

The FINDPUSHABLEJOBSREVERSE algorithm takes list of migratable jobs and the task sets allocated to each core. It finds the list of migratable jobs in active period from the end of active period ( $C_i.LMJReverse[]$ ). With this list, it computes the stolenAP as summation of WCET of migratable jobs. It returns the  $C_i.LMJReverse[]$  and stolenAP.

### 5.2.2.2 Motivating Example

Consider a task set  $T$  consisting of six periodic hard real-time tasks with temporal parameters shown in table 5.2. The tasks  $T_0$ ,  $T_1$ ,  $T_2$  and  $T_4$  are allocated to core  $C_1$  and tasks  $T_3$  and  $T_5$  are allocated to core  $C_2$  for scheduling. Consider scheduling of jobs with AET as 70% of estimated WCET. At  $t=0$ , the scheduler calls RECOMMENDER. RECOMMENDER computes AP as 88 and NID as 12 units. Assuming  $NID < SDT$  (40), it calls FINDPUSHABLEJOBS and PROCRASTINATE which could not find sufficient idle duration for shutting down the core. It computes extra idle duration required for shutting down

---

**Algorithm 12: FINDPUSHABLEJOBSREVERSE**


---

**Input:**  $\forall_{i=1 \text{ to } m} C_i.MQ[], T_{C_i}, \text{ReqdExtraID}$ 
**Output:**  $C_S.LMJReverse[]$ 

```

1 stolenAP=0
2 foreach job  $J_i \in C_S.AP$  in reverse order of release time do till
   migratable
3   foreach core  $C_T \in Lcores[]$  in decreasing order of utilization do
4     compute  $C_T.slack$  between  $J_i.r$  &  $J_i.d$ 
5     if ( $C_T.slack \geq J_i.WCET$ ) then
6        $C_S.LMJReverse[] \leftarrow J_i$ 
7       stolenAP +=  $J_i.WCET$  and break
8     end
9   end
10 end
11 return  $C_i.LMJReverse[], stolenAP$ 

```

---

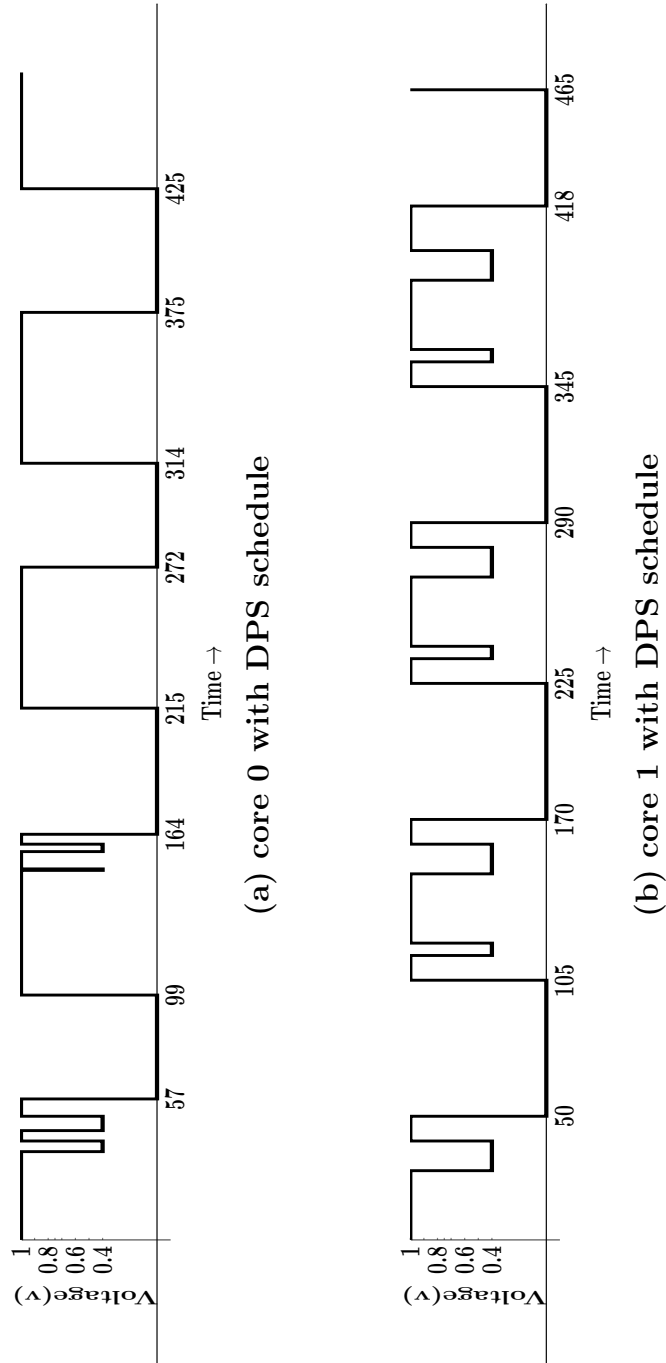
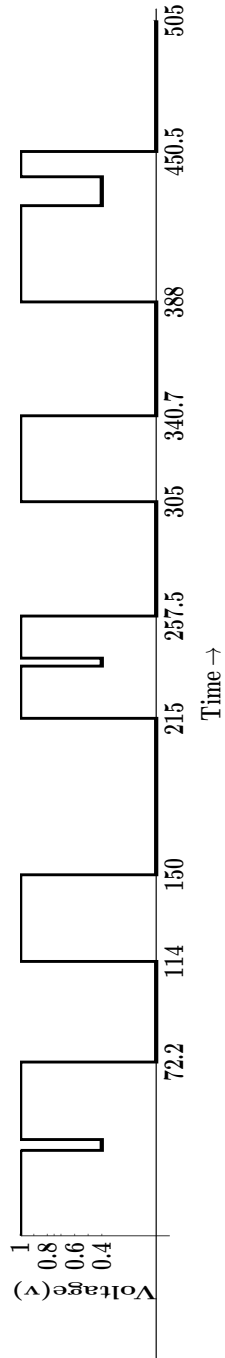
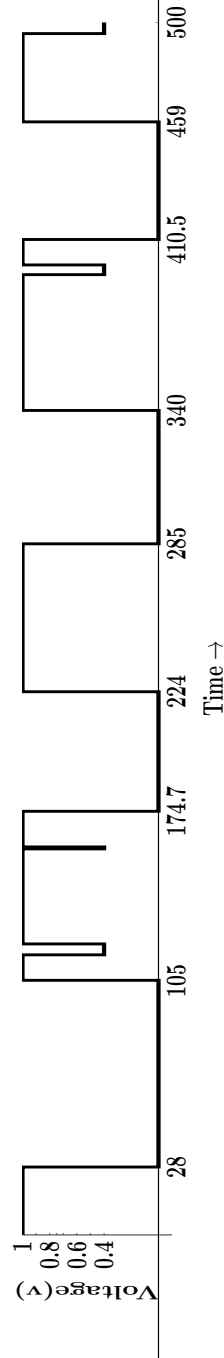


Figure 5.16: Core State Transitions

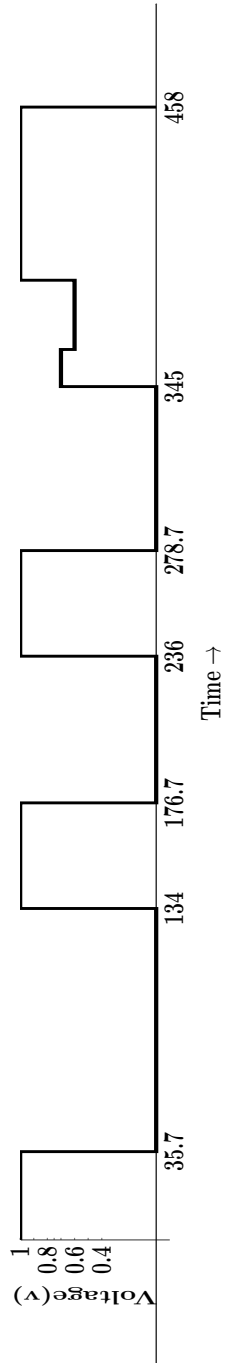


(c) core 0 with OASIS schedule

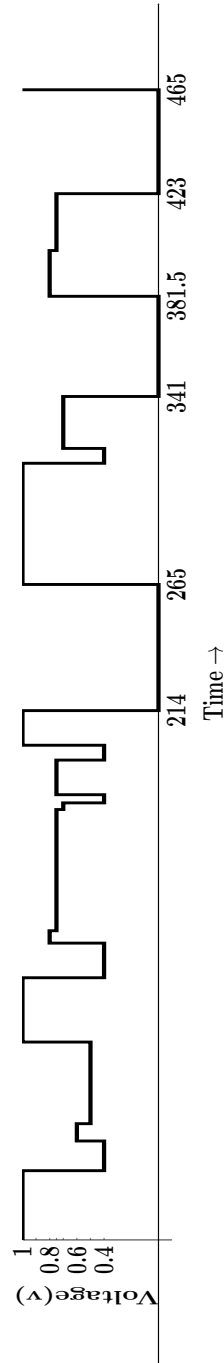


(d) core 1 with OASIS schedule

Figure 5.16: Core State Transitions



(e) core 0 with HANdT schedule



(f) core 1 with HANdT schedule

Figure 5.16: Core State Transitions



Table 5.2: Task set 4

Taskno	Period	WCET
$T_0$	40	6
$T_1$	50	10
$T_2$	60	15
$T_3$	40	15
$T_4$	100	20
$T_5$	120	25

as  $\text{ReqdExtraID} = 28$ . It calls `FINDPUSHABLEJOBSREVERSE` and passes  $\text{ReqdExtraID}$ . `FINDPUSHABLEJOBSREVERSE` finds the list of pushable jobs as  $J_{0,1}, J_{0,2}, J_{1,1}, J_{2,2}$  and returns it with corresponding target cores and  $\text{stolenAP}$  to `RECOMMENDER`. Since  $\text{stolenAP}$  is more than the  $\text{ReqdExtraID}$ , `RECOMMENDER` pushes the jobs to corresponding target cores, computes AP, NID period and sets the scaled voltage to maximum. It return the scaled voltage value to the scheduler `HANDT`. `HANDT` executes the highest priority job at set voltage. The resultant core state transitions using Dynamic Procrastination Scheduler (DPS) (described in Chapter 4), OASIS (described in Section 5.2.1) and `HANDT` for one hyper-period are shown in Figure 5.16 a, b, c, d and e, f respectively. The voltage levels for shutdown and idle states of core is represented by 0v and 0.4v respectively on y axis. The voltage levels for active state of core ranging between 0.4v to 1v is shown on y axis. It can be observed that the shutdown duration in schedule with `HANDT` has increased and idle duration has reduced followed by the schedules with DPS and OASIS.

### 5.2.2.3 Analysis of HANDT Algorithm

#### Schedulability analysis

**Theorem 5.2.5** *For a periodic task set with implicit deadlines having soft affinity, HANDT produces a valid and feasible schedule if there exist one by any dynamic priority driven scheduling approach.*

**Proof:** Since each core in HANDT follows semi-static task allocation and dynamic priority scheduling, a periodic task set with implicit deadlines can be feasibly scheduled on the core to which it is allocated such that it follows equation (4.3). The necessary and sufficient condition for migrating a job  $J$  is to have sufficient slack in the core to which it is allocated. The slack is computed by equation 5.2. The slowdown and shutdown decisions in HANDT make sure it maintains the schedulability bound after following RECOMMENDER algorithm and  $P^3$  policy if each core follows equation (4.16) after voltage scaling and equation (5.1) after migration and procrastination. With reference to figure 4.6, RECOMMENDER sets the value for supply voltage and operating frequency at the beginning of AP ( $_{current}AP_{begin}$ ). It is decided to slowdown till the beginning of next active period ( $_{next}AP_{begin}$ ) only if the PI formed by pushing and procrastinating is less than the shutdown threshold. The scheduler also decides to slow down on completion of job if it finishes earlier than its WCET. Thus it does not affect the schedulability. Equation (4.17) shows that the mechanism of extending lower priority job execution time by taking care of higher priority job deadlines will not affect higher priority jobs in  $_{current}AP$ . Thus the tasks in source core remains schedulable after voltage and frequency scaling. The tasks are migrated to the target

cores only if it follows equation 5.1. Thus the task set remains schedulable after following RECOMMENDER algorithm. Theorem 5.2.1 shows that the task set remains schedulable after applying  $P^3$  policy.

### Correctness analysis

**Theorem 5.2.6** *The overall shutdown duration produced by the HANDT scheduler is greater than or equal to the one produced by DPS and EDF scheduling algorithms.*

**Proof:** Let  $S_{DPS}$ ,  $S_{EDF}$ ,  $S_{OASIS}$  and  $S_{HANDT}$  be the schedules produced by DPS, EDF, OASIS and HANDT scheduling algorithms. Let  $SD_{DPS}$ ,  $SD_{EDF}$ ,  $SD_{OASIS}$  and  $SD_{HANDT}$  be the total shutdown duration in  $S_{DPS}$ ,  $S_{EDF}$ ,  $S_{OASIS}$  and  $S_{HANDT}$  respectively. DPS algorithm procrastinates the execution of future jobs and merges the intermediate idle intervals within same core which results into longer shutdown duration. The migration of jobs from other cores and procrastination of upcoming jobs by HANDT increases the procrastinated idle duration to get better overall shutdown duration. Thus

$$SD_{HANDT} \geq SD_{DPS} \quad (5.18)$$

The pull migration in OASIS may give chance to shutdown other cores. The slowing down of core in HANDT reduces the idle duration and thus reduces the chance to perform pull migration. This reduces the number of shutdown states which inherently reduces the shut down overhead (SDO).

$$SD_{OASIS} \geq SD_{HANDT} \quad (5.19)$$

and

$$SDO_{\text{HANDT}} \leq SDO_{\text{OASIS}} \quad (5.20)$$

Eq.(4.7), (5.18) and (5.19)  $\implies$

$$SD_{\text{OASIS}} \geq SD_{\text{HANDT}} \geq SD_{\text{DPS}} \geq SD_{\text{EDF}} \quad (5.21)$$

**Theorem 5.2.7** *The overall idle duration produced by HANDT scheduler is less than or equal to the one produced by DPS and OASIS scheduling algorithms.*

**Proof:** Let  $ID_{\text{DPS}}$ ,  $ID_{\text{OASIS}}$  and  $ID_{\text{HANDT}}$  be the total idle duration and  $AD_{\text{DPS}}$ ,  $AD_{\text{OASIS}}$  and  $AD_{\text{HANDT}}$  be the total active duration in  $S_{\text{DPS}}$ ,  $S_{\text{OASIS}}$  and  $S_{\text{HANDT}}$  respectively. When the idle duration is less than shutdown threshold, DPS algorithm keeps the core idle. Whereas OASIS migrates the jobs from other cores without changing the overall active period. This may increase the overall shutdown duration in OASIS schedules. Thus

$$AD_{\text{OASIS}} = AD_{\text{DPS}} \quad (5.22)$$

Equations 5.21 and 5.22  $\implies$

$$ID_{\text{OASIS}} \leq ID_{\text{DPS}} \quad (5.23)$$

HANDT also migrates the jobs from other cores when idle duration is less than shutdown threshold and no job is ready in that core's ready queue. If

there are ready jobs and next idle interval is less than the shutdown threshold, HANDT executes the jobs at reduced voltage and frequency. Thus converts the unused idle interval into active duration. Thus

$$AD_{\text{HANDT}} \geq AD_{\text{OASIS}} \quad (5.24)$$

and

$$ID_{\text{HANDT}} \leq ID_{\text{OASIS}} \quad (5.25)$$

Equations (5.23) and (5.25)  $\implies$

$$ID_{\text{HANDT}} \leq ID_{\text{OASIS}} \leq ID_{\text{DPS}} \quad (5.26)$$

Equations (5.22) and (5.24)  $\implies$

$$AD_{\text{HANDT}} \geq AD_{\text{OASIS}} = AD_{\text{DPS}} \quad (5.27)$$

**Theorem 5.2.8** *The schedule produced by the HANDT scheduler offers better energy saving compared to any dynamic priority scheduling algorithm.*

**Proof:** Let  $SE_{\text{DPS}}$ ,  $SE_{\text{OASIS}}$  and  $SE_{\text{HANDT}}$  be the total static energy in  $S_{\text{DPS}}$ ,  $S_{\text{OASIS}}$  and  $S_{\text{HANDT}}$  respectively. Eq.(5.21)  $\implies$

$$SE_{\text{OASIS}} \leq SE_{\text{HANDT}} \leq SE_{\text{DPS}} \quad (5.28)$$

Let  $DE_{\text{DPS}}$ ,  $DE_{\text{OASIS}}$  and  $DE_{\text{HANDT}}$  be the total dynamic energy in  $S_{\text{DPS}}$ ,  $S_{\text{OASIS}}$  and  $S_{\text{HANDT}}$  respectively. Though Eq.(5.27) implies that the active

period of HANDT schedule is more than OASIS and DPS, the jobs are executed at reduced voltage and frequency. Thus

$$DE_{\text{HANDT}} \leq DE_{\text{OASIS}} \leq DE_{\text{DPS}} \quad (5.29)$$

Let  $ENERGY_{\text{DPS}}$ ,  $ENERGY_{\text{OASIS}}$  and  $ENERGY_{\text{HANDT}}$  be the overall energy while scheduling  $S_{\text{DPS}}$ ,  $S_{\text{OASIS}}$  and  $S_{\text{HANDT}}$  respectively. As dynamic energy consumption is multi-fold higher than static energy consumption, the overall energy of HANDT is much lesser than other algorithms, i.e.

$$ENERGY_{\text{HANDT}} \leq ENERGY_{\text{OASIS}} \leq ENERGY_{\text{DPS}} \quad (5.30)$$

## Complexity analysis

HANDT is a task level dynamic priority scheduling algorithm with events as (i) beginning of active period (ii) job(s) arrival (iii) core wakeup and (iv) job completion. The run time complexity of HANDT is  $\text{Max}\{A_1, A_2, A_3, A_4\}$  where  $A_1, A_2, A_3$  and  $A_4$  are the run time complexities of events i, ii, iii and iv respectively.

**Run time complexity of Event (i):**  $A_1 = A_5 + \text{constant time for selection of job from ready queue for execution where } A_5 \text{ is run time complexity of RECOMMENDER algorithm.}$

**Run time complexity of Event (ii):**  $A_2 = \text{time required for updations,}$

maintaining priority queue +  $A_6$  + constant time for selection of highest priority job, where  $A_6$  is run time complexity of SELECTOR algorithm.

$$A_2 = N \log N + A_6 + C$$

from section (4.3.2.4),  $A_6 = N \log N + L \log L$

$$\text{So } A_2 = N \log N + N \log N + L \log L + C = 2N \log N + L \log L + C$$

**Run time complexity of Event (iii):**  $A_3$  = selection of job from ready queue and execution is a constant time operation = C.

**Run time complexity of Event (iv):**  $A_4 = \text{Max} \{ (\text{time for } P^3), (A_2) \}$

From section (5.2.1.3), run time complexity of  $P^3$  policy is  $PN^2 + L \log L$

$$\text{So } A_4 = \text{Max} \{ (PN^2 + L \log L), (2N \log N + L \log L) \}$$

$$A_4 = PN^2 + L \log L$$

**Run time complexity  $A_5$ :** With reference to RECOMMENDER algorithm,  $A_5$  = time required for computing AP and NID + Max{(lines 2 to 4), (lines 5 to 30)}

Lines 2 to 4: setting scaled voltage for source core is constant time operation.

Lines 5 to 30: constant time for computing EAP +  $A_7$  +  $A_8$  + constant time for computing NSAP + Max{(lines 10 to 13), (lines 14 to 23)} + Max{(24 to 26), (lines 27 to 39)}

$$= C + A_7 + A_8 + C + \text{Max}\{C, (A_9 + C)\} + \text{Max}\{C, C\} = A_7 + A_8 + A_9,$$

where  $A_7$ ,  $A_8$  and  $A_9$  are the run time complexities of FINDPUSHABLEJOBS, PROCRASTINATE and FINDPUSHABLEJOBSREVERSE algorithms.

$$\text{So } A_5 = N \log N + \text{Max}\{(C), (A_7 + A_8 + A_9)\} = N \log N + A_7 + A_8 + A_9$$

So  $A_1 = N \log N + A_7 + A_8 + A_9$

From section (5.2.1.3),  $A_7 = NP + PN^2$  and  $A_8 = L \log L$

So  $A_1 = N \log N + NP + PN^2 + L \log L + A_9$

**Run time complexity  $A_9$ :**

With reference to FINDPUSHABLEJOBSREVERSE algorithm,

Line 2: sorting jobs based on release time -  $N \log N$ .

Line 3: Finding utilization of P cores, each with N tasks and sorting based on utilization -  $PN + P \log P$ .

Line 4: The slack for job j is computed by considering the remaining execution time of running job, WCET of ready jobs and WCET of migrated/future jobs that arrives before the deadline of job j. It is seen experimentally that the number of jobs considered for slack calculation varies from 1 to 2N. Thus run time complexity for slack calculation is  $2N + \text{constant } C$ .

Lines 5 to 8: On finding sufficient slack the job is added to the list of migratable jobs and stolenAP is computed in constant time.

Lines 2 to 10: The work has restricted to check the slack for  $2*N$  migratable jobs.

Lines 3 to 9: The slack is checked in P cores.

So,  $A_9 = N \log N + PN + P \log P + P * ( 2N + C ) + C = N \log N + PN + PN^2$

So  $A_1 = N \log N + PN + PN^2 + L \log L + N \log N + PN + PN^2 = 2(PN + PN^2 + N \log N) + L \log L$

Thus the run time complexity of HANDT =  $Max\{A_1, A_2, A_3, A_4\} =$



$$\text{Max}\{(2(PN + PN^2 + N \log N) + L \log L), (2N \log N + L \log L), (C), (PN^2 + L \log L)\}$$

$$= 2(PN + PN^2 + N \log N) + L \log L$$

In asymptotic notation it is  $\mathcal{O}(PN + PN^2 + N \log N + L \log L) = PN^2$ .

#### 5.2.2.4 Experimental Evaluation

**Experimental setup:** The experimentation is conducted using the task model described in Chapter 3. The framework SMART described in Chapter 3 is used to find the schedule and measure the performance of HANDT algorithm in comparison with seminal algorithms. The task sets are scheduled using ccEDF, DPS, DPVFS, OASIS and HANDT algorithms. The framework evaluates these scheduling algorithms based on shutdown duration, static, dynamic, inactive duration and total energy consumptions.

**Experimental results:** Figures [5.17], [5.18], [5.19] and [5.20] show the effect of DP and migration techniques over DVFS technique by comparing HANDT with ccEDF. It also shows the effect of DVFS and migration techniques over DP technique by comparing HANDT with DPS. It shows the effect of migration technique over the combined effect of DP and DVFS by comparing HANDT with DPVFS.

Figure 5.17 shows the shutdown duration for different utilizations. Migration in HANDT increases the shutdown duration than DPVFS but scaled active duration in HANDT sometimes reduces the chances of shutting down the core. On an average, HANDT scheduler reduces shutdown duration by 2.76%

over DPS and increase by 0.84% over DPVFS schedules.

Figure 5.18 shows the static energy consumption for different utilizations. Irrespective of the utilization, ccEDF consumes constant static energy since it do not shutdown the core. On an average, HANDT algorithm reduces static energy by 50% over ccEDF. Since static energy is inversely proportional to the shutdown duration, on an average, HANDT increases the static energy by 3% over DPS and reduces by 0.83% over DPVFS algorithm.

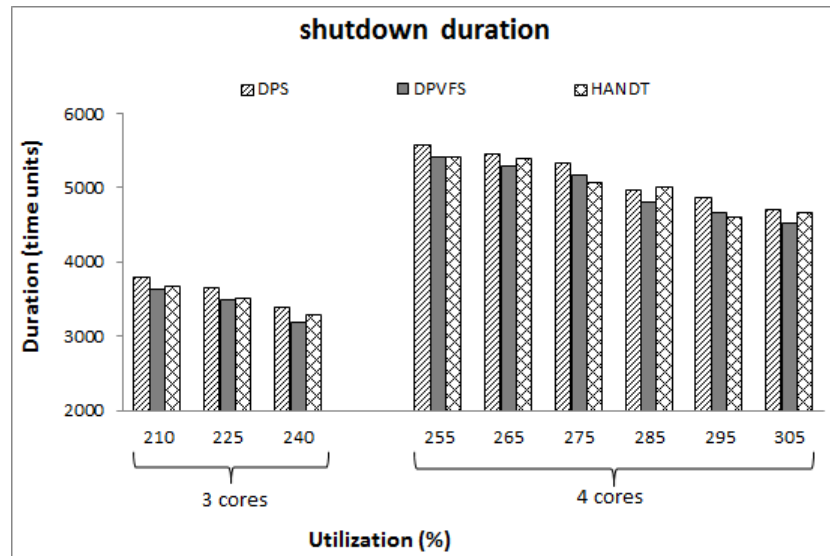


Figure 5.17: Shutdown duration for different utilizations

Figure 5.19 shows the dynamic energy consumption for different utilizations. On an average, HANDT increases the dynamic energy consumption by 5.5% over ccEDF and reduces by 6% and 1.46% over DPS and DPVFS algorithms respectively.

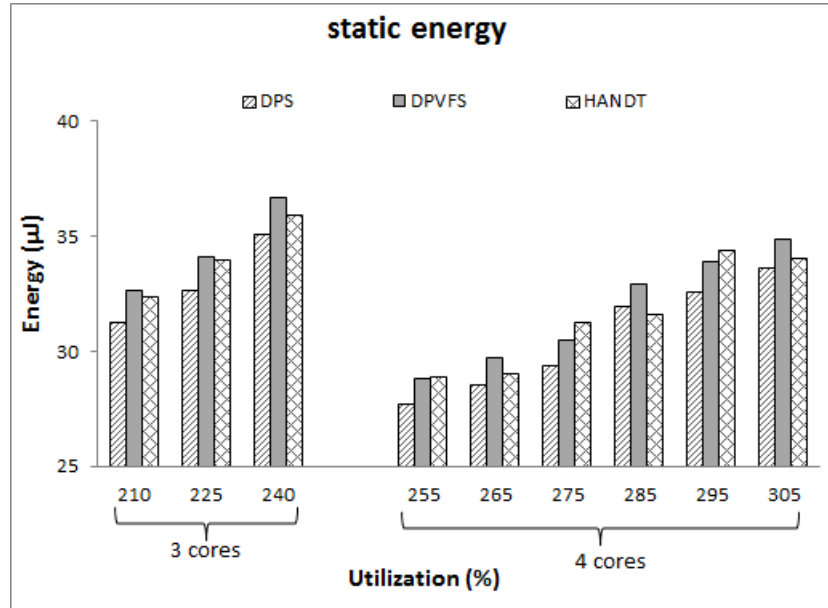


Figure 5.18: Static energy consumption per unit for different utilizations

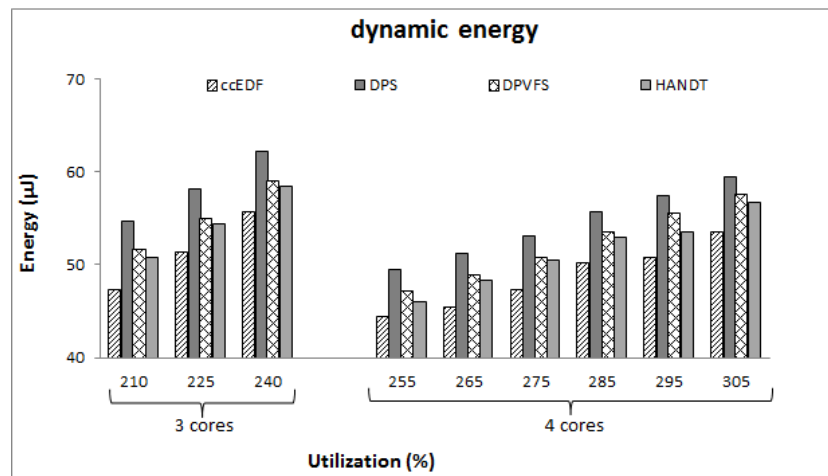


Figure 5.19: Dynamic energy consumption per unit for different utilizations

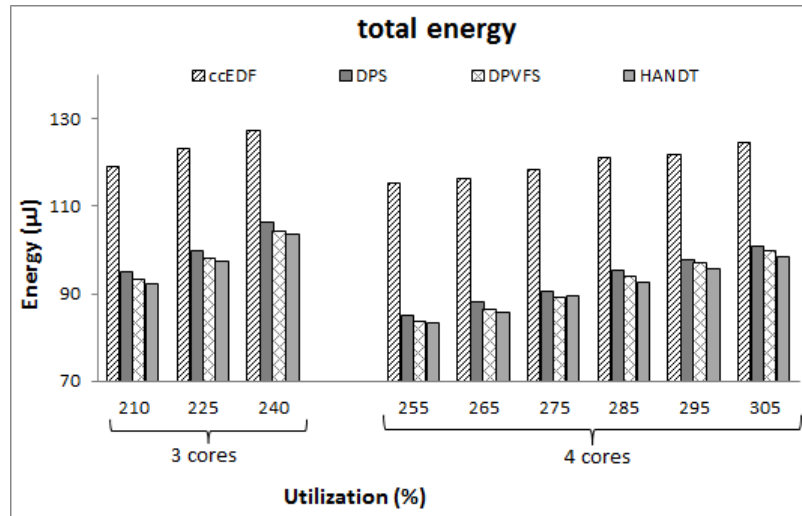


Figure 5.20: Total energy consumption per unit for different utilizations

Figure 5.20 shows the total energy consumption for different utilizations. On an average, HANDT reduces the total energy consumption by 23%, 2.4% and 1% over ccEDF, DPS and DPVFS algorithms respectively.

Figures 5.21, 5.22, 5.23, 5.24 and 5.25 shows the combined effect of DP, DVFS and migration techniques over DP with migration by comparing HANDT with OASIS algorithm.

Figure 5.21 shows the shutdown duration for different utilizations. The scaled active duration in HANDT sometimes reduces the chances of shutting down the core. On an average, HANDT scheduler reduces shutdown duration by 4.8% over OASIS schedule.

Figure 5.22 shows the static energy consumption for different utilizations. Since static energy is inversely proportional to the shutdown duration, on

an average, HANDT increases the static energy consumption by 5.33% over OASIS algorithm.

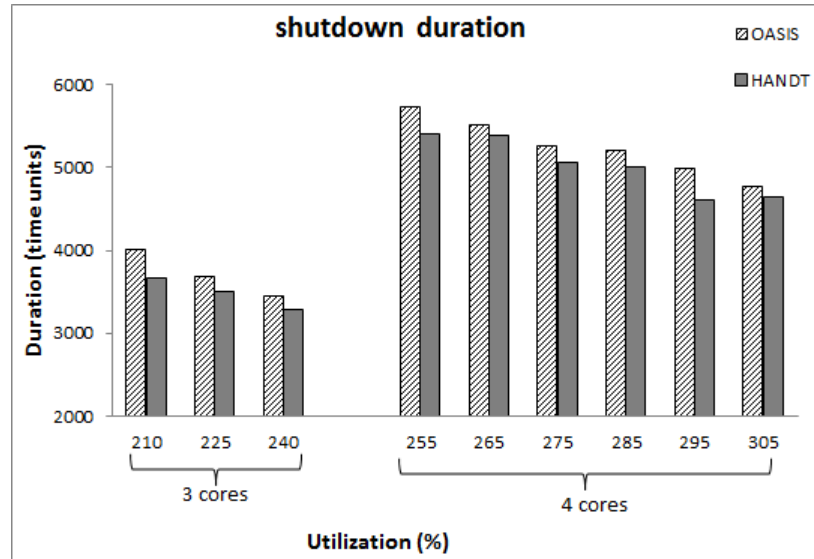


Figure 5.21: Shutdown duration for different utilizations

Figure 5.23 shows the active duration for different utilizations. In HANDT, to utilize the inactive idle duration, the jobs are executed at reduced voltage and frequency. On an average, the active duration in HANDT schedule increases and thus the voltage/frequency scales down by 6.73% over the schedules produced by DPS and OASIS algorithms respectively.

Figure 5.24 shows the dynamic energy consumption for different utilizations. Since dynamic energy depends on voltage and frequency during active and idle duration, on an average, HANDT scheduler reduces the dynamic energy consumption by 5.3% over OASIS algorithm.

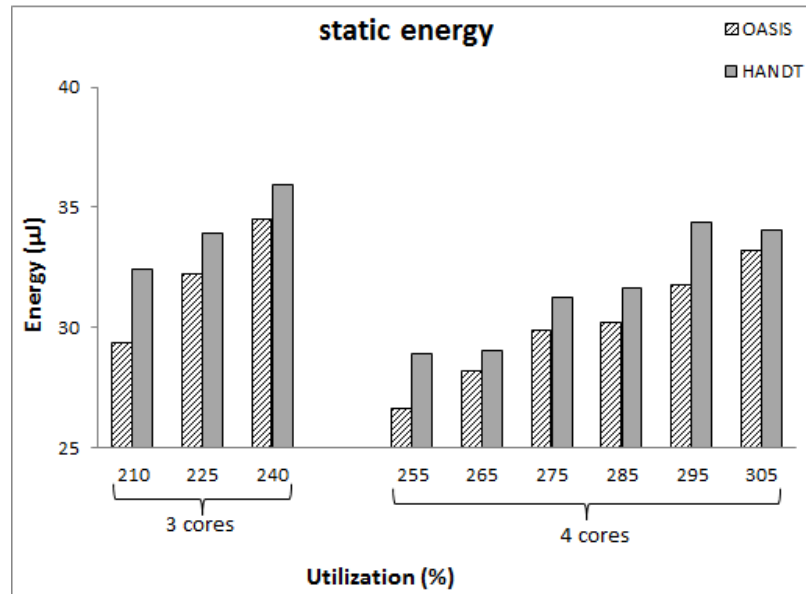


Figure 5.22: Static energy consumption per unit for different utilizations

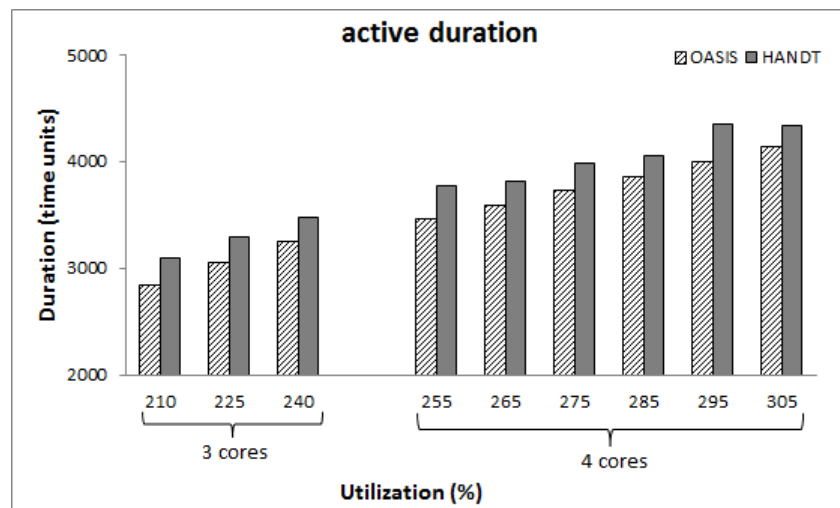


Figure 5.23: Active duration for different utilizations

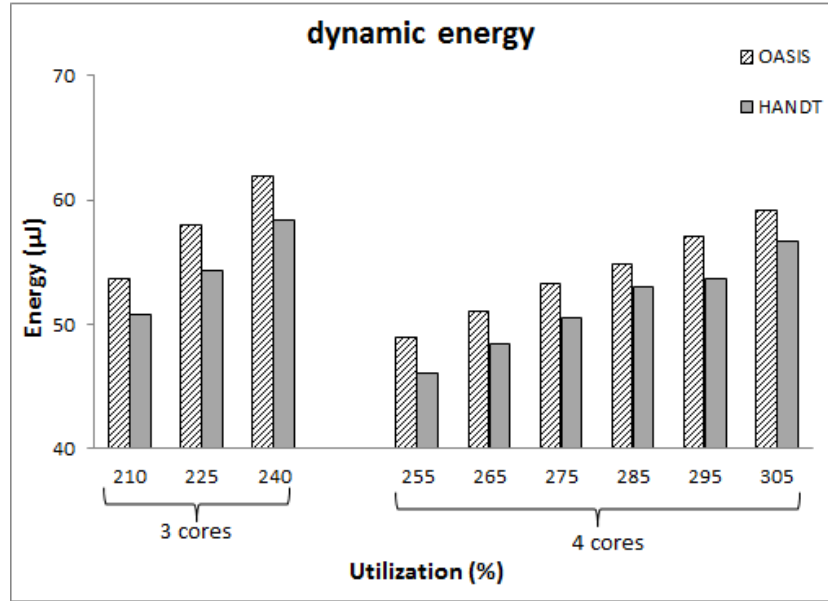


Figure 5.24: Dynamic energy consumption per unit for different utilizations

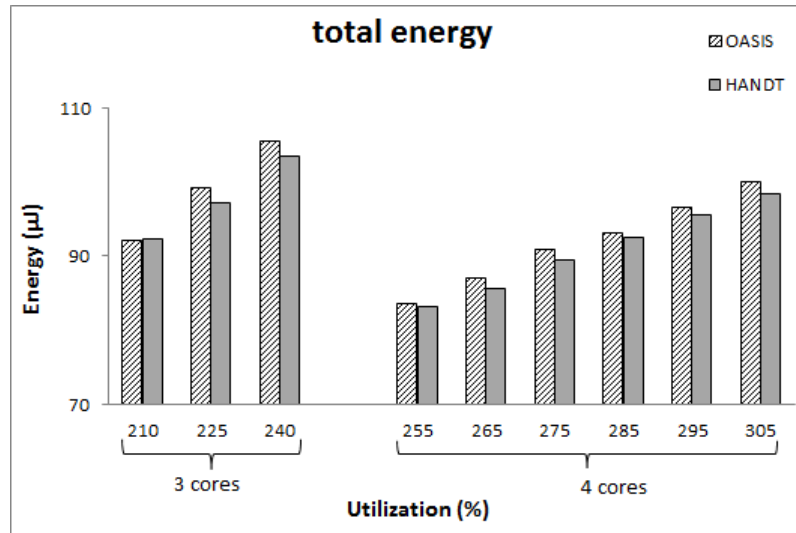


Figure 5.25: Total energy consumption per unit for different utilizations

Figure 5.25 shows the total energy consumption for different utilizations. On an average, HANDT reduces the total energy consumption by 1.3% over OASIS algorithm.

**Conclusion:** Along with the primary constraint of timeliness on real time tasks, HANDT scheduler also minimizes overall energy consumption while scheduling by reducing the static and dynamic energy consumption over DPS, ccEDF, DPVFS and OASIS algorithms. It is applicable to the MC system supporting various operating modes and having discrete levels of voltages and frequencies.

### 5.3 Summary

In this chapter **OASIS** - an **O**ptimal **s**tatic energy **S**cheduler with **m**igration and dynamic **p**rocrastination scheduler, **HANDT** - Hare **A**ND **T**ortoise scheduler are described, analyzed and evaluated in detail. These algorithms are designed for making homogeneous MC-HRTS energy efficient that follows semi-partitioning allocation of tasks having soft affinity. OASIS scheduler uses  $P^3$  policy to dynamically decide push or pull migration. By adopting Migration with Dynamic Procrastination technique, it increases the chances of shutting down the core. Migration also helped HANDT to utilize the unused idle duration of the core by executing the jobs from other cores and making chances of shutdown in other core. Thus reduces static and dynamic energy consumption in the schedule. HANDT uses Migration along with DP and DVFS techniques for optimizing energy consumption.



HANDT scheduler optimizes the static and dynamic energy consumption using RECOMMENDER algorithm and  $P^3$  policy which recommends the choice between appropriate voltage/frequency scaling and procrastination. If the system has prolonged inactive time, HANDT scheduler adopt hare policy of finishing all the existing jobs at the earliest to shut down for a longer duration. To increase the shutdown duration it migrates some of the jobs to active cores. Thus reduces both static and dynamic energy consumption. When the system has only short bursts of inactive periods, it follows tortoise policy of reducing the speed. The unproductive idle time is utilized by slowing down the core or by migrating and executing the jobs from other cores so as to shutdown the other cores if possible. This results in saving static and dynamic energy consumption. HANDT is applicable to the MC system that supports discrete voltage and frequency levels and multiple modes of core like active, idle and shutdown. Since both the schedulers are dynamic, optimal shutdown duration / optimal voltage scaling can be obtained. The detailed analysis of these algorithms is done based on schedulability, correctness and complexity. OASIS and HANDT can produce a valid and feasible schedule on procrastination and voltage/frequency scaling. The run time complexity of OASIS algorithm is  $\mathcal{O}(PN^2 + L \log L)$  and that of HANDT is  $\mathcal{O}(PN^2)$ . The schedules produced using OASIS and HANDT reduces energy consumption in comparison with various scheduling algorithms - EDF, EDFSD, Conventional DPS, proposed DPS, ccEDF and DPVFS. It is observed that OASIS algorithm produces schedule with least energy consumption compared to other algorithms designed for system that supports migration and has no DVFS support. HANDT algorithm produces

schedule with least energy than other algorithms designed for systems that support migration and has multiple voltage and frequency levels.

# Chapter 6

## Conclusion and Future Directions

This chapter consolidates remarks on conducted research work, its limitations and future directions in energy efficient scheduling techniques. It also summarizes the results obtained and observations made after performing the experimentation with seminal and proposed scheduling algorithms.

### 6.1 Summary of results

Motivated by the fact that performance requirement demands increase in number of cores at the cost of high energy consumption which causes fatal effects in real time systems. This thesis addresses energy minimization in MC system for hard real time tasks. This work also addressed energy consumption during process execution and scheduling at operating system level. In order to understand various techniques used for energy minimization while maintaining the timeliness constraint in real time systems, a survey was carried out. The major components of energy consumption identified from the literature are static energy due to leakage current and dynamic energy due to switching current. The technology advancement has made static

energy equally an important component contributing to the overall energy consumption as dynamic energy. During idle state of core, it consumes static and dynamic energy without performing any significant activity. Any schedule can be energy efficient if the core remains idle for minimum time and shutdown for maximum time. The increase in shutdown duration effectively reduces static and dynamic energy consumption. Widely used method in literature for static energy savings is dynamic procrastination. This thesis has addressed maximization of shutdown duration at task allocation as well as task scheduling stages of MC system. A modified version of First Fit Bin Packing is developed as an energy efficient task allocation method named MFFBP. For testing the results, EDF algorithm is used for scheduling with FFBP and MFFBP allocation, allowing to shutdown the core whenever possible. Experimental results show that MFFEDF\_SD reduces the static and dynamic energy by 13.43% and 8.42% respectively over FFEDF\_SD algorithm. At scheduling stage, DPS algorithm is designed to increase the shutdown duration for static energy saving. For producing the test schedule and analysis purpose, a simulator named SMART is developed. To observe the effect of DPS algorithm over static procrastination method, MFFSTATICPRO and MFFDPS algorithms are simulated and compared using SMART. Experimental result shows that MFFDPS reduces the static energy consumption by 18.3% and overall energy consumption by 10.7% over MFFSTATICPRO algorithm. To see the combined effect of MFFBP task allocation and DPS algorithm, MFFDPS is compared with conventional DPS method and DPS with FF allocation. Experimental result shows that MFFDPS reduces the static energy by 10.7%, 5.33%, dynamic energy by 4%,

1.8% and total energy by 6.2%, 3.1% over ConvDPS and FFDPS algorithms respectively. DPS method is designed for tasks having hard affinity towards the core of the MC system having multiple modes of core operating with single frequency.

This thesis addressed static and dynamic energy minimization using a combination of DP and DVFS techniques. DPVFS algorithm is designed to dominate procrastination over voltage/frequency scaling. It tries to reduce the idle time by first converting into shutdown. The idle time is further reduced by applying DVFS. To observe the combined effect of procrastination and voltage/frequency scaling, DPVFS algorithm is compared with pure DVFS method- ccEDF and pure procrastination method - DPS. Experimental results show that DPVFS reduces the static energy by 84.54% over ccEDF and increases marginally by 1.45% over DPS. DPVFS increases dynamic energy marginally by 0.38% over ccEDF and saves 32.7% over DPS. DPVFS reduces total energy by 33.2% and 18.8% over ccEDF and DPS respectively. DPVFS is specifically designed for MC system which support multiple voltage and frequency levels and have tasks with hard affinity towards the cores.

The shutdown duration can also be increased by merging the idle intervals spread across the cores. This is possible if the migration of jobs is allowed. A solution is provided by this thesis is OASIS algorithm. In OASIS, we applied migration along with procrastination to increase the shutdown duration and to minimize the idle duration. To see the effect of migration over procrastination OASIS is compared with conventional procrastination

and DPS methods. Experimental results show that OASIS reduces static and overall energy by 8%, 0.7% and by 3.88%, 1.2% over ConvDPS and DPS methods respectively.

The leftover idle intervals by OASIS can be reduced by applying DVFS to reduce dynamic energy consumption. This thesis addresses static and dynamic energy savings using migration along with DP and DVFS techniques. HANDT algorithm is designed to first apply migration with DP. Whenever this do not produce effective shutdown duration, it applies DVFS or migration. To see the effect of combined effect of DP, DVFS and migration techniques, HANDT is compared with DPS, DPVFS and OASIS. Experimental results shows that HANDT reduces static energy by 0.83%, 5.33% over DPVFS and OASIS algorithms respectively. HANDT reduces dynamic energy by 6%, 1.46% and 5.3% over DPS, DPVFS and OASIS algorithm respectively. HANDT reduces overall energy by 2.4%, 1% and 1.3% over DPS, DPVFS and OASIS algorithms respectively.

To schedule the real time tasks with minimum energy consumption in MC system supporting different technologies, a ready reckoner is provided in table 6.1 to choose the appropriate scheduling algorithm.

## 6.2 Future scope

This thesis considered independent, processor bound, periodic hard real time tasks with implicit deadlines. We plan to extend this work by considering

Table 6.1: Ready reckoner for choice of algorithm

Techniques Supported		Affinity	
DVFS	Shutdown	Hard	Soft
×	×	RM, Util < 80%	
		EDF, Util $\geq$ 80%	
✓	×	ccEDF	
×	✓	DPS	OASIS
✓	✓	DPVFS	HANDT

mixed task set with sporadic and aperiodic tasks along with periodic. To execute sporadic job along with the existing periodic jobs, the sporadic job has to pass through the acceptance test so as to avoid deferred execution of existing periodic job and already accepted sporadic jobs. Sporadic tasks can be treated same as periodic tasks if the inter arrival time between consecutive jobs are periodic. In practice, occurrence of sporadic jobs are much lesser than its periodic arrivals. This challenges investigating the number of cores required, number of cores in active and shutdown state etc. The acceptance test can be eased by profiling the sporadic jobs based on past history. The aperiodic job can be executed along with periodic and sporadic jobs by using slack stealing methods to improve the response time of it.

We are also in the process of exploring job level dependency and sharing among jobs of various tasks. The dependency and sharing among jobs will be explored with respect to resource bound tasks where we intend to consider memory, memory mapped I/O and serial I/O resources. The future scope also includes considering preemptive and non-preemptive resources by adding

block time due to resource dependency. Our proposed methods aimed at energy savings for tightly coupled homogeneous MC systems. We intend to extend this work for tightly coupled heterogeneous MC systems. The concepts of energy saving through DVFS, DP and migration can be further extended for distributed MP [homogeneous and heterogeneous] environment. The challenges are availability and fault tolerance in real time system to manage loosely coupled processing elements with data and resource dependency. In Internet of Things (IOT), we intend to extend these schedulers for mixed criticality real time systems where criticality and resource sharing are equally important as priority of the task.



# Publications

$C_1$ : DPS: A Dynamic Procrastination Scheduler for Multi-core/  
Multi-processor Hard Real Time Systems, Shubhangi K. Gawali, Biju K.  
Raveendran, In IEEE International Conference on Control, Decision and  
Information Technologies (CoDIT), pages 286 - 291, 2016.

$J_1$ : DPVFS: A Dynamic Procrastination cum DVFS Scheduler for Multicore  
Hard Real Time Systems, Shubhangi K. Gawali, Biju K. Raveendran,  
International Journal of Embedded system (IJES), Inderscience publication,  
2017.

# A brief biography of the candidate

Shubhangi is a Ph.D. candidate with particular interests in Operating systems and Real time systems. She is a lecturer in CS/IS department of BITS Pilani, K. K. Birla Goa Campus. Prior to enrolling at BITS Pilani University, she worked for eight years as a lecturer in engineering institutes. She holds a Masters degree in Computer Engineering from NMIMS University, Mumbai and Bachelors degree in Computer Engineering from the University of Mumbai. When at home, she enjoys to sketch pictures in company of her daughter Spandana.

# Brief biography of supervisors

Dr. Biju K. Raveendran is currently serving as Assistant Professor in the Department of Computer Science and Information Systems, BITS Pilani K. K. Birla Goa campus, Goa, India. He received his Ph.D. degree from BITS Pilani, Pilani campus, Rajasthan in the year 2009. He heads the Computer Center Unit at Goa campus which is responsible for the central networking and computing facilities of the campus. His research areas include Energy Efficient Multi-core/Many-core Real-time Scheduling and Memory Architecture for Embedded Systems etc. He is a recipient of Microsoft young faculty award in year 2009. He is also a recipient of Best Faculty Award by BITSAA in the year 2013. He is actively involved in collaborative projects with industries like Microsoft, Aditya Birla Group etc.

Prof. Bharat M. Deshpande is heading the Department of Computer Science and Information Systems, Bits Pilani K. K. Birla Goa Campus, Goa, India. He received his Ph.D. degree from IIT Mumbai in the year 1998. After which for a year he worked as postdoctoral fellow in Department of Atomic Energy. His research interests are in areas of Complexity Theory, Parallel Algorithms, and Data Mining. Over the years he has supervised numerous masters and doctoral students. He has many national and international publications to his credit.

# Bibliography

- [1] O. S. Unsal and I. Koren. System-level power-aware design techniques in real-time systems. volume 91, pages 1055–1069, 2003.
- [2] N. K. Jha. Low power system scheduling and synthesis. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. *IEEE/ACM Digest of Technical Papers*, pages 259–263, 2001.
- [3] L. Luo. *Designing Energy and User Efficient Interactions with Mobile Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2008.
- [4] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real time embedded systems. In *41st IEEE Design Automation Conference*, pages 275–280, 2004.
- [5] P. Padmanabhan and S. Kang. Real-time dynamic voltage scaling for low-power embedded operating systems. volume 35, pages 89–102, 2001.
- [6] H. Aydin, R. Melhem, D. Mosse, and Pedro. Power-aware scheduling for periodic real-time tasks. In *IEEE Transactions on Computers*, pages 584–600, 2004.
- [7] C. Yang, J. Chen, and T. Kuo. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In *IEEE Design, Automation and Test in Europe*, pages 468–473, 2005.

- 
- [8] R. Jejurikar and R. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *42nd IEEE Design Automation Conference*, pages 111–116, 2005.
  - [9] Y. Lee, K. Reddy, and C. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *15th IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, pages 105–112, 2003.
  - [10] L. Niu and G. Quan. Reducing both dynamic and leakage energy consumption for hard real time systems. In *CASES: ACM International conference on Compilers, architecture and synthesis for embedded systems*, pages 140–148, 2009.
  - [11] V. Legout, M. Jan, and L. Pautet. A scheduling algorithm to reduce the static energy consumption of multiprocessor real-time systems. In *21st ACM International conference on Real-Time Networks and Systems (RTNS)*, pages 99–108, 2013.
  - [12] S. Pagani and J. J. Chen. Energy efficiency analysis for the single frequency approximation (sfa) scheme. In *IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 82–91, 2013.
  - [13] J. Gu and G. Qu. Incorporating temperature-leakage interdependency into dynamic voltage scaling for real-time systems. In *IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, pages 289–296, 2013.

- 
- [14] M. Khaleel and M. Zhu. Energy-efficient task scheduling and consolidation algorithm for workflow jobs in cloud. volume 13, pages 268–284. Inderscience Publishers, 2016.
- [15] L. Geunsik, M. Changwoo, and E. Y. Load-balancing for improving user responsiveness on multicore embedded systems. In *Proceedings of the Linux Symposium*, pages 25–33, 2012.
- [16] C. Keng-Mao, T. Chun-Wei, C. Yi-Shiuan, and Y. Chu-Sing. A high performance load balance strategy for real-time multicore systems. Hindawi Publishing Corporation, 2014.
- [17] G. Buttazzo. *Hard Real-Time Computing Systems*. Springer, 2011.
- [18] Jane Liu. *Real-Time Systems*. Pearson Education, 5 edition, 2004.
- [19] R. Nassiffe, E. Camponogara, G. Lima, and D. Moss. Optimising qos in adaptive real-time systems with energy constraint varying cpu frequency. pages 368–379. Inderscience Publishers, 2016.
- [20] L. Niu and G. Quan. Reducing both dynamic and leakage energy consumption for hard real-time systems. In *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, pages 140–148, 2004.
- [21] P. Kocanda and A. Kos. Static and dynamic energy losses vs. temperature in different cmos technologies. In *22nd International Conference Mixed Design of Integrated Circuits Systems (MIXDES)*, pages 446–449, 2015.

- 
- [22] N. Srinivasan, N. Prakash, D. Shalakh, D. Sivaranjani, G. Swetha, and B. Bala. Power reduction by clock gating technique. volume 21, pages 631–635. Elseiver, 2015.
- [23] V. Devdas and H. Aydin. On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications. In *IEEE Transactions on Computers*, pages 31–44, 2012.
- [24] G. Chen, K. Huang, and A. Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination. volume 13, pages 1–21, 2014.
- [25] J. Chen and T. Kuo. Procrastination for leakage-aware rate monotonic scheduling on a dynamic voltage scaling processor. In *ACM SIGPLAN on language, compilers and tool support form embedded systems*, pages 153–162, 2006.
- [26] J. Chen and T. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 289–294, 2007.
- [27] E. Seo, S. Kim, S. Park, and J. Lee. Dynamic alteration schemes of real-time schedules for i/o device energy efficiency. volume 10, pages 1–32, 2011.
- [28] J. Lee and A. Shrivastava. Pica: Processor idle cycle aggregation for energy-efficient embedded systems. volume 11, pages 1–27, 2012.

- 
- [29] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo. Energy-aware scheduling for real-time systems: A survey. volume 15, pages 1–34, 2016.
- [30] A. Mishra and A. Tripathi. Energy efficient voltage scheduling for multi-core processors with software controlled dynamic voltage scaling. pages 3456–3466. Elsevier, 2014.
- [31] M. Bambagini, M. Bertogna, and G. Buttazzo. On the effectiveness of energy-aware real-time scheduling algorithms on single-core platforms. In *IEEE Proceedings of the Emerging Technology and Factory Automation (ETFA)*, pages 1–8, 2014.
- [32] O. Zapata and P. Alvarez. Edf and rm multiprocessor scheduling algorithms: Survey and performance evaluation. Technical report, <http://delta.cs.cinvestav.mx/pmejiamultitechreport.pdf>, 2005.
- [33] K. Kedar, R. Harini, S. Abhik, and M. Frank. Policies for migration of real-time tasks in embedded multi-core systems. In *Real-time systems symposium*, pages 17–20, 2009.
- [34] R. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. volume 43, 2011.
- [35] S. Baruah. Techniques for multiprocessor global schedulability analysis. volume 22, page 14, 2007.
- [36] A. Björn and B. Konstantinos. Sporadic multiprocessor scheduling with few preemptions. In *Euromicro Conference on Real-Time Systems (ECRTS)*, volume 8, pages 243–252, 2008.



- 
- [37] A. Björn, B. Konstantinos, and B. Sanjoy. Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In *Real-Time Systems Symposium*, pages 385–394, 2008.
- [38] K. Shinpei and Y. Nobuyuki. Portioned edf-based scheduling on multiprocessors. In *Proceedings of the 8th ACM international conference on Embedded software*, pages 139–148, 2008.
- [39] K. Shinpei, Y. Nobuyuki, and I. Yutaka. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 249–258, 2009.
- [40] A. Pillai and T. Isha. Ec-a: A task allocation algorithm for energy minimization in multiprocessor systems. volume 8, pages 254–260, 2014.
- [41] Jr. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for np-hard problems. chapter Approximation Algorithms for Bin Packing: A Survey, pages 46–93. PWS Publishing Co., 1997.
- [42] M. Bambagini, J. Lelli, G. Buttazzo, and G. Lipari. On the energy-aware partitioning of real-time tasks on homogeneous multi-processor systems. In *4th Annual International Conference on Energy Aware Computing Systems and Applications (ICEAC)*, pages 69–74, 2013.
- [43] W. Mao. Best k fit packing. pages 265–270, 1993.
- [44] T. AlEnawy and H. Aydin. Energy-aware task allocation for rate monotonic scheduling. In *11th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 213–223, 2005.

- 
- [45] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS)*, pages 95–105, 2001.
- [46] J.J. Chen, T. Kuo, and C. Yang. Profit-driven uniprocessor scheduling with energy and timing constraints. In *Proceedings of ACM Symposium on Applied Computing (SAC)*, pages 834–840, 2004.
- [47] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pages 197–202, 1998.
- [48] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 374–382, 1995.
- [49] J. J. Chen, T. Kuo, and H. Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. In *Workshop on Algorithms and Data Structures*, pages 338–349, 2005.
- [50] P. Shiv and P. Deo. A hybrid immune genetic algorithm for scheduling in computational grid. volume 6. Inderscience publishers, 2014.
- [51] P. Shiv, T. Vibhu, and R. Manojkumar. An elitist non-dominated sorting bat algorithm nsbat-ii for multi-objective optimization of phthalic antride reactor. volume 7, pages 200–315. Inderscience publishers, 2016.

- 
- [52] T. Vibhu, P. Shiv, and R. Manojkumar. Optimized on-line control of mma polymerization using fast multi-objective de. pages 1–8. Taylor and Francis Online, 2016.
- [53] Y. Zhang, X. Hu, and D. Chen. Task scheduling and voltage selection for energy minimization. In *Proceedings of the 39th Annual Design Automation Conference (DAC)*, pages 183–188, 2002.
- [54] W. Shieh and C. Pong. Energy and transition-aware runtime task scheduling for multicore processors. volume 73, pages 1225–1238. Academic Press, Inc., 2013.
- [55] X. Chen, X. Zheng, K. Hyungjun, G. Paul, Hu. Jiang, M. Kishinevsky, U. Ogras, and R. Ayoub. Dynamic voltage and frequency scaling for shared resources in multicore processor designs. In *Proceedings of the 50th Annual Design Automation Conference (DAC)*, volume 114, pages 1–7, 2013.
- [56] N. Min-Allah, H. Hussain, S. Khan, and A. Zomaya. Power efficient rate monotonic scheduling for multi-core systems. volume 72, pages 48–57. Academic Press, Inc., 2012.
- [57] X. Zhu, C. He, K. Li, and X. Qin. Adaptive energy-efficient scheduling for real-time tasks on dvs-enabled heterogeneous clusters. volume 72, pages 751–763. Academic Press, Inc., 2012.
- [58] R. Bergamaschi, H. Guoling, A. Buyuktosunoglu, H. Patel, I. Nair, G. Dittmann, G. Janssen, N. Dhanwada, Zhigang Hu, P. Bose, and J. Darringer. Exploring power management in multi-core systems. In

- 
- Asia and South Pacific Design Automation Conference*, pages 708–713, 2008.
- [59] K. Woonseok Kim, K. Jihong, and L. Sang. Preemption-aware dynamic voltage scaling in hard real-time systems. In *IEEE Proceedings of International Symposium on Low Power Electronics and Design*, pages 393–398, 2004.
- [60] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. In *IEEE Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2003.
- [61] D. Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 35–40, 2004.
- [62] B. Goel and S. McKee. A methodology for modeling dynamic and static power consumption for multicore processors. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 273–282, 2016.
- [63] C. Y. Yang, J. J. Chen, L. Thiele, and T. W. Kuo. Energy-efficient real-time task scheduling with temperature-dependent leakage. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 9–14, 2010.
- [64] W. Liao, L. He, and K. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. volume 24, pages 1042–1053, 2005.

- 
- [65] Y. Liu, R. Dick, L. Shang, and H. Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *Design, Automation Test in Europe Conference Exhibition*, pages 1–6, 2007.
- [66] D. Meisner, B. Gold, and T. Wenisch. Powernap: Eliminating server idle power. volume 37, 2009.
- [67] M. Awan and S. Petters. Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems. In *23rd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 92–101, 2011.
- [68] K. Huang, L. Santinelli, J. Chen, L. Thiele, and G. Buttazzo. Periodic power management schemes for real-time event streams. In *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) with 28th Chinese Control Conference*, pages 6224–6231, 2009.
- [69] A. Rowe, K. Lakshmanan, H. Zhu, and R. Rajkumar. Rate-harmonized scheduling and its applicability to energy management. *IEEE Transactions on Industrial Informatics*, 6(3):265–275, 2010.
- [70] S. DSouza, A. Bhat, and R. Rajkumar. Sleep scheduling for energy-savings in multi-core processors. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 226–236, 2016.
- [71] C. Fu, Y. Zhao, M. Li, and C. J. Xue. Maximizing common idle time on multi-core processors with shared memory. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 900–903, 2015.
- [72] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. volume 3, 2007.

- 
- [73] A. Gabriel, V. Sameer, B. Rémi, S. Gilles, B. Pascal, T. Lionel, C. Everton, and M. Fernando. Evaluating the impact of task migration in multi-processor systems-on-chip. In *Proceedings of the 23rd symposium on Integrated circuits and system design*, pages 73–78, 2010.
- [74] S. Abhik, M. Frank, R. Harini, and M. Sibin. Push-assisted migration of real-time tasks in multi-core processors. In *ACM Sigplan Notices*, volume 44, pages 80–89, 2009.
- [75] J.J. Chen, H. Heng-Ruey, C. Kai-Hsiang, Y. Chia-Lin, P. Ai-Chun, and K. Tei-Wei. Multiprocessor energy-efficient scheduling with task migration considerations. In *Proceedings of 16th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 101–108, 2004.
- [76] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of edf on multiprocessor platforms. In *17th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 209–218, 2005.

# Index

Affinity, 38

    hard affinity, 38

    soft affinity, 38

Aggressive Speed Reduction (AGR), 31

Bin Packing (BP), 30

    Best Fit BP (BFBP), 30

    First Fit BP (FFBP), 8, 30, 57

    Modified FFBP (MFFBP), 45, 56, 58

    Next Fit BP (NFBP), 30

    Worst Fit BP (WFBP), 30

critical speed, 34, 36

Cycle Conserving EDF (ccEDF), 26

dynamic energy, 2, 51

Dynamic Reclaiming Algorithm (DRA), 31

Global scheme, 106

Look Ahead EDF (LAEDF), 26

Migration, 38

    pull migration, 39, 109

    push migration, 38, 109

Partitioned scheme, 106

Procrastination, 3

    Dynamic Procrastination (DP), 3, 6, 34, 66

    Procrastinated Idle Duration (PID), 71

    Static Procrastination, 66

Semi-partitioned scheme, 106

source core, 107, 109

static energy, 2, 51

target core, 109

Voltage/Frequency Scaling (VFS), 2

    Dynamic VFS (DVFS), 2, 6, 31

    Static VFS (SVFS), 26