Scheduling of real time embedded systems for resource and energy minimization by voltage scaling

Naveen Babu Anne
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
https://digitalscholarship.unlv.edu/rtds/1819

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
Thesis Approval
The Graduate College
University of Nevada, Las Vegas

August 10, 2005

The Thesis prepared by

Naveen Babu Anne

Entitled

"Scheduling of Real Time Embedded Systems for Resource and
Energy Minimization by Voltage Scaling"

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

[Signatures]

Examination Committee Chair

Dean of the Graduate College

[Signatures]

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ABSTRACT

Scheduling of Real Time Embedded Systems for Resource and Energy Minimization by Voltage Scaling

by

Naveen Babu Anne

Dr. Venkatesan Muthukumar, Examination Committee Chair
Assistant Professor of Electrical and Computer Engineering
University of Nevada, Las Vegas

The aspects of real-time embedded computing are explored with the focus on novel real-time scheduling policies, which would be appropriate for low-power devices. To consider real-time deadlines with pre-emptive scheduling policies will require the investigation of intelligent scheduling heuristics. These aspects for various other RTES models like Multiple processor system, Dynamic Voltage Scaling and Dynamic scheduling are the focus of this thesis. Deadline based scheduling of task graphs representative of real time systems is performed on a multiprocessor system.

A set of aperiodic, dependent tasks in the form of a task graph are taken as the input and all the required task parameters are calculated. All the tasks are then partitioned into two or more clusters allowing them to be run at different voltages. Each cluster, thus voltage scaled results in the overall minimization of the power utilized by the system. With the mapping of each task to a particular voltage done, the tasks are scheduled on a multiprocessor system consisting of processors that can run at different voltages and frequencies, in such a way that all the timing constraints are satisfied.
### TABLE OF CONTENTS

ABSTRACT ............................................................................................................................... iii

LIST OF FIGURES .................................................................................................................. vi

LIST OF TABLES ..................................................................................................................... vii

ACKNOWLEDGEMENTS ....................................................................................................... viii

CHAPTER 1 INTRODUCTION ................................................................................................ 1
Real Time Embedded System (RTES) .................................................................................. 1
Real Time Applications ......................................................................................................... 1
Representing a Real Time Embedded System ..................................................................... 2
Design of Real Time Embedded System ............................................................................ 2
Performance characterization of RTES ............................................................................... 3

CHAPTER 2 SCHEDULING IN REAL TIME SYSTEMS ...................................................... 7
Definitions ............................................................................................................................... 7
The Need for Scheduling: Scheduler and Scheduling ....................................................... 8
Commonly used approaches to Real-Time Scheduling ..................................................... 9
  Clock Driven Scheduling ............................................................................................... 10
  Weighted Round-Robin Approach .............................................................................. 10
  Priority Driven Scheduling Approach ..................................................................... 11
Scheduling Policies in Real Time Systems: Classification ............................................. 12
  Off-line/On-line (Static/ Dynamic) ........................................................................... 12
  Preemptive/Non-preemptive ...................................................................................... 13
  Centralized/Distributed .............................................................................................. 14

CHAPTER 3 REVIEW OF EXISTING LITERATURE ............................................................. 15
Scheduling Algorithms: The Big Picture .......................................................................... 15
Real Time Task Model .......................................................................................................... 16
Voltage Scaling and Energy Minimization ......................................................................... 16
Standard Task Graph (STG) ............................................................................................... 19
Description of the Existing Scheduling Algorithms .......................................................... 19

CHAPTER 4 DESCRIPTION OF THE SCHEDULING ALGORITHMS ................................ 33
Mathematical Modeling of the Scheduling Problem .......................................................... 33
Algorithm Description ....................................................................................................... 34
  The Scheduling after Scaling (SaS) Algorithm ............................................................ 34
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling before Scaling Algorithm (SbS)</td>
<td>37</td>
</tr>
<tr>
<td>Probability Based Scheduling (PbS) Algorithm</td>
<td>40</td>
</tr>
<tr>
<td>Energy - Probability Based Scheduling (E-PbS) Algorithm</td>
<td>46</td>
</tr>
<tr>
<td>An Example illustrating the algorithms</td>
<td>48</td>
</tr>
<tr>
<td>SaS</td>
<td>49</td>
</tr>
<tr>
<td>SbS</td>
<td>51</td>
</tr>
<tr>
<td>Probability based Scheduling</td>
<td>54</td>
</tr>
<tr>
<td>E-PbS</td>
<td>57</td>
</tr>
</tbody>
</table>

**CHAPTER 5 RESULTS, CONCLUSION AND FUTURE WORK** ............................................. 60

- Results .................................................................................................................. 60
- Conclusion ................................................................................................................ 61
- Benchmarks ............................................................................................................... 61
- Assumptions made .................................................................................................... 62
- Simulation Results ................................................................................................. 62
- Directions for future work ..................................................................................... 78

**REFERENCES** .......................................................................................................... 79

**VITA** ....................................................................................................................... 83
LIST OF FIGURES

Fig.1. Design flow of real time embedded system ............................................................... 4
Fig.2. Classification of RTES Scheduling Algorithms ....................................................... 12
Fig.3. Classification of Scheduling Algorithms into Fixed and Dynamic Priority Algos. 15
Fig.4. Standard Task Graph ............................................................................................... 18
Fig.5. Rate Monotonic Schedule of tasks T1, T2, T3 ......................................................... 20
Fig.6. Deadline Monotonic Schedule of Tasks T1, T2, T3 in 8.1.2 ..................................... 24
Fig.7. Earliest Deadline First Schedule of Tasks T1, T2, T3 in 8.2.1 ................................... 25
Fig.8. Least Slack Time First Scheduling of Tasks T1, T2, T3 in 8.2.2 ............................. 32
Fig.9. Standard Task Graph Example .............................................................................. 48
Fig.10. Scheduling after Scaling ....................................................................................... 51
Fig.11. Scheduling before Scaling .................................................................................... 52
Fig.12. Scaled processors in SbS algorithm ....................................................................... 53
Fig.13. ASAP and ALAP schedules for 10.stg ................................................................. 55
Fig.14. Final PbS Schedule for 10.stg ............................................................................. 58
Fig.15. Final E-PbS Schedule for 10.stg ......................................................................... 59
Fig.16. Comparison chart depicting No. of processors vs. No. of tasks ......................... 64
Fig.17. Division of total number of processors for SaS Algorithm by Voltage ............... 64
Fig.18. Division of total number of processors for SbS Algorithm by Voltage ............... 65
Fig.19. Division of total number of processors for PbS Algorithm by Voltage ............... 65
Fig.20. Division of total number of processors for E-PbS Algorithm by Voltage .......... 66
Fig.21. Energy Comparison of Number of Tasks vs. Energy Consumed ....................... 68
Fig.22. Energy consumed per each voltage in SaS Algorithm .......................................... 69
Fig.23. Energy consumed per each voltage in SbS Algorithm .......................................... 69
Fig.24. Energy consumed per each voltage in PbS Algorithm .......................................... 70
Fig.25. Energy consumed per each voltage in E-PbS Algorithm ...................................... 70
Fig.26. Comparison of Processor utilization vs. Number of tasks ................................... 73
Fig.27. Comparison of Processor utilization at 5.0V for each algorithm ......................... 73
Fig.28. Comparison of Processor utilization at 3.3V for each algorithm ......................... 74
Fig.29. Comparison of Processor utilization at 2.4V for each algorithm ......................... 74
Fig.30. Average Utilization of the processors expressed in percentage ......................... 75
Fig.31. Comparison chart for Energy consumed per processor ........................................ 75
Fig.32. Bar graph showing the energy per processor consumed by each algorithm ......... 76
Fig.33. Graph showing the product of Energy and Utilization ........................................ 77
Fig.34. Bar graph depicting the efficiency of each algorithm ......................................... 77
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Tabular form of STG Graph shown in Figure 9</td>
<td>49</td>
</tr>
<tr>
<td>Table 2</td>
<td>Voltage scaling of each node</td>
<td>50</td>
</tr>
<tr>
<td>Table 3</td>
<td>Scaling in SbS algorithm</td>
<td>53</td>
</tr>
<tr>
<td>Table 4</td>
<td>Table showing ASAP and ALAP start times and Mobility of each task</td>
<td>54</td>
</tr>
<tr>
<td>Table 5</td>
<td>Probabilities for task# 1 from ASAPbegin till ALAPend</td>
<td>56</td>
</tr>
<tr>
<td>Table 6</td>
<td>Distribution Graph for Time Step-1</td>
<td>56</td>
</tr>
<tr>
<td>Table 7</td>
<td>Table showing number of processors required</td>
<td>63</td>
</tr>
<tr>
<td>Table 8</td>
<td>Energy consumed by the processors while implementing the algorithms</td>
<td>67</td>
</tr>
<tr>
<td>Table 9</td>
<td>Average utilization of the processors</td>
<td>71</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Dr. Venkatesan Muthukumar whose insights and suggestions for possible approaches helped find a solution that led to success. He has been a continuous source of inspiration. I benefited a lot from working with him and as his teaching assistant.

My special thanks go to Ms. Shruti Patil who gave my research work a valuable impetus by providing crucial directions for the solution of the problem I had been working on.

I wish to thank Dr. Ajoy K. Datta and Dr. Henry Selvaraj for sparing their valuable time and consenting for being a part of my examination committee. A wonderful thanks goes to Dr. Emma Regentova who apart from being in my examination committee, has been very cordial and encouraging at every stage of my masters program. I would especially like to thank my father Dr. Anne Subbarao and my mother Dr. Madala Ramadevi who encouraged me to take up graduate studies.

Finally, I would like to extend thanks to all my colleagues and faculty members in the Digital Design Group for creating a conducive environment for research and brainstorming. To all friends and other people close to me goes out my biggest "Thank you" of all.
CHAPTER 1

INTRODUCTION

1.1 Real Time Embedded System (RTES)

Definition: Any system where a timely response by that system to external stimuli is vital is known as a Real Time System.

Definition of Components: Components here refer to individual processing elements like processors.

A real time embedded system is a system whose behavior depends on the accuracy of its logic as well as its timing. It is a collection of components working sequentially or in parallel, each with a specialized functionality that is designed to perform a specific operation. These components interact with each other either by exchanging information or by sharing resources. The degree of this interaction may vary in time which can be of the order of microseconds to hours and days.

Real Time Operating System (RTOS) is an operating system that works in a real time computing environment. The real time computing environment refers to the situation which has power, energy and timing constraints.

1.2 Real Time Applications

Typical real time embedded system applications are: washing machines, central
heating systems, automatic gate keepers, cahier machines at parking areas. More complex and sophisticated ones are: flight control avionics, process control in industries, nuclear plant monitoring, scientific experiment guidance in laboratories, air traffic control, robotics, remote exploration of underwater, space and high risk environments, surgical operation and patient monitoring, command and control in defense and virtual reality systems.

1.3 Representing a Real Time Embedded System

The use of Computer-aided-Design (CAD) tools in RTES design allows for streamlining their design and facilitates component reuse for future designs. Component reuse implies saving and using the design templates or designs for future RTES models which results in reducing the costs and the time-to-market of the product. CAD tools provide performance accurate modeling and simulations thereby reducing the time-to-market of the real time application.

A Real Time Embedded System can be modeled as a graph consisting of one or more nodes and edges connecting the nodes. A node represents a task within the RTES and the edge between two nodes represents the dependency or exchange of information involved between two tasks. Depending on the nature and utility of the system, the tasks of a system can be made to run either on a single processor or a multiprocessor system.

1.4 Design of Real Time Embedded System

Figure 1 shows the design methodology of a real time embedded system. The design
of a RTES is mainly consists of the following areas:

- System Specification
- Hardware-software partitioning
- Software design
- Hardware Synthesis
- Software – Hardware Interface Synthesis
- Hardware implementation

The hardware-software partitioning is performed based on the requirements and the specifications of the system. The hardware synthesis of the system gives the optimized design in terms of number of resources required. In the software design part, it is made sure that the system meets all its real time constraints like power and timing. The system is then scheduled to be executed on the resources selected. The scheduling of the system task is done by the scheduler which is controlled by the operating system used. Finally, the whole real time system is implemented in the actual hardware.

1.5 Performance characterization of RTES

A Real Time Embedded System is characterized by its ability to meet the timing constraints. Power and resource constraints are the performance metrics for the RTES. The power consumed and the resources (e.g. Processors) used indicate the efficiency of the system. An efficient RTES consumes minimum power and minimum resources. The main problem in the design and the research is directed towards minimizing the power.
There are two components of power i.e., static component and dynamic component. Static power comes from circuit design techniques that include voltage bias generators and any DC paths through active devices. Leakage power is due primarily to sub-threshold leakage currents that result from reduced threshold voltages that prevent transistors from turning completely off. Dynamic power is dissipated when a device is switching. Switching power comes from charging and discharging capacitive loads. Short circuit power is dissipated due to the current that conducts when both the n-channel and p-channel transistors are momentarily on at the same time and is dependent on the switching frequency, input slew rate and the difference between the operating and
threshold voltages. Standby power is dissipated when a device is not switching. The power consumption in a CMOS circuit is given by the following formula

\[ P_{\text{total}} = P_{\text{static}} + P_{\text{short}} + C_{\text{sw}} \cdot f \cdot V_{\text{dd}}^2 + P_{\text{glitching}} \]

The static power, \( P_{\text{static}} \), is the power consumed through leakage currents and it occurs even when the circuit does not operate. This power is very small for CMOS circuits, almost negligible. \( P_{\text{short}} \) occurs with every gate output switching, when two output transistors of a CMOS gate are open in the same time. With a good design and technology this power can be kept under 10% of the dynamic power. The third term in the equation is the switching power and it is dependent on the clock frequency, the supply voltage and the switching capacitance. The last term is the power dissipation due to glitching. In this paper the minimization of power due to switching has been prioritized. The term \( C_{\text{sw}} V_{\text{dd}}^2 \) states that power consumption is dependent on frequency of operation, switching capacitance, which depends on the size of the load (wire capacitance, output capacitance of driver, and input capacitance of the driven cells), and the square of the operating voltage. It is clear from the equation that reducing the supply voltage, clock frequency, switching capacitance or switching activity in the circuit reduces the dynamic component of the power. A variety of optimization methods targeting each of these four factors have been explored.

This thesis proposes four different algorithms for power minimization in RTES. These algorithms proceed by reducing the voltages of operation of the processors and distributing tasks running on the processors. This involves scheduling the tasks on the processors and scaling the voltage of operation which is explained in Chapter 4. Chapter 2 introduces the reader to the definitions and terminologies used in this thesis. An
extensive survey of the literature on scheduling algorithms aimed at power and energy minimization in RTES is discussed in Chapter 3. The results and conclusion are explained in Chapter 5 with directions for future work.
CHAPTER 2

SCHEDULING IN REAL TIME SYSTEMS

2.1 Definitions

Task Set: A real time application is specified by means of a set of tasks.

Task: A real time task is an executable entity of work which at a minimum, is characterized by a worst case execution time and a time constraint.

There are three types of real-time tasks: periodic, aperiodic, and sporadic.

Periodic Tasks: Periodic tasks are real-time tasks which are activated (released) regularly at a fixed period. The time constraint for a periodic task is a deadline 'd' that can be less than, equal to or greater than the period.

Aperiodic Tasks: Aperiodic tasks are real-time tasks which are activated irregularly at some unknown and possibly unbounded rate. The time constraint is usually a deadline 'd'.

Sporadic Tasks: Sporadic tasks are real-time tasks which are activated irregularly with some known bounded rate. The bounded rate is characterized by a minimum interarrival period, that is, a minimum interval of time between two successive activations. The time constraint is usually a deadline 'd'.

Deadline: A deadline 'd' is a point in time by which the task must complete its execution. Usually a deadline 'd' is an absolute time. Sometimes, 'd' is also referred to as a relative deadline. The deadline can be hard, soft, or firm.
Hard Deadline: A hard deadline means that it is vital for the safety of the system that this deadline is always met.

Soft Deadline: A soft deadline means that it is desirable to finish executing the task by the deadline, but no catastrophe occurs if there is a late completion.

Firm Deadline: A firm deadline means that a task should complete by the deadline, or not execute at all. There is no value to completing the task after its deadline.

Release Time: The release time of a task is the instant of time at which the task becomes available for execution. The task can be scheduled and executed at any time at or after its release time whenever its data and control dependency conditions are met.

Relative Deadline: The maximum allowable response time of a job is its relative deadline.

Absolute Deadline: This is equal to the task’s release time plus its relative deadline.

Period: The difference in time between the arrivals of two consecutive instances of a periodic task is fixed and is referred to as the period of the task.

Hyperperiod: The Hyperperiod of a set of tasks is the time interval of a fixed length which is equal to the least common multiple of the periods of all the tasks.

Execution Time: This is the computation time of the task.

Slack: Slack of a task is the difference of its execution time and the deadline.

Precedence constraint: If a certain task can execute only after the completion of its predecessor task(s), then such a task is called as precedence constrained task.

2.2 The Need for Scheduling: Scheduler and Scheduling

Scheduling a real time system is necessary to satisfy the following requirements:
- All the tasks in the system meet their timing constraints and system as a unit operates successfully.
- Allocate resources (processors) to all the tasks.
- Optimize resources, power and time utilization of the system.

Tasks are scheduled according to a chosen set of scheduling algorithms and resource access-control protocols. The module which implements these algorithms is called a scheduler. Specifically, the scheduler assigns the processor to tasks, or equivalently, assigns tasks to processors. A task is scheduled in a time interval on a processor if the processor is assigned to the task and hence the task executes on the processor, in the interval. The total amount of time assigned to a task according to a schedule is the total length of all the time intervals during which the task is scheduled on some processor.

A schedule means an assignment of all the tasks in the system on the available processors produced by the scheduler. A scheduler produces a valid schedule if it satisfies the following conditions:

- Every processor is assigned to at most one task at any time.
- Every task is assigned at most one processor at any time.
- No task is scheduled before its release time.
- Depending on the scheduling algorithms used, the total amount of processor time assigned to every task is equal to its maximum or actual execution time.
- All the precedence and resource usage constraints are satisfied.

2.3 Commonly used approaches to Real-Time Scheduling

The subsequent sections will briefly explain clock-driven, weighted round-robin
approaches and will give a detail explanation of priority driven scheduling.

2.3.1 Clock Driven Scheduling [1]

As the name implies, when scheduling is clock-driven (also called time driven), decisions on what tasks can execute at what times are made at specific time instants. These instants are chosen a priori before the system begins execution. Typically, in a system that uses clock-driven scheduling; all the parameters of hard real-time tasks are fixed and known. A schedule of the tasks is computed off-line and is stored for use at run time. The scheduler schedules the jobs according to this schedule at each scheduling decision time. In this way, scheduling overhead during run time is minimized.

2.3.2 Weighted Round-Robin Approach [1]

The round-robin approach is commonly used for scheduling applications which have to share the processors time for completing their execution. When tasks are scheduled on a round-robin basis, every task joins a first-in first-out queue when it becomes ready for execution. The task at the head of the queue executes for at most one time slice. (A time slice is the basic granule of time that is allocated to tasks. In a time-shared environment, a time slice is typically in the order of tens of milliseconds.) If the task does not complete by the end of the time slice, it is preempted and placed at the end of the queue to wait for its next turn. When there are \( n \) ready tasks in the queue, each task gets one time slice every \( n \) time slices, that is, every round. Because the length of the time slice is relatively short, the execution of every task begins almost immediately after it becomes ready. In essence, each task gets \( 1/n \)th share of the processor when there are \( n \) tasks ready for execution. This is why the round-robin algorithm is also called the processor sharing algorithm.
By giving each task a fraction of the processor, a round-robin scheduler delays the completion of every task. If it is used to schedule precedence constrained tasks, the response time of a chain of tasks can be very large. For this reason, the weighted round-robin algorithm is not suitable for scheduling such tasks. On the other hand, a successor task may be able to incrementally consume what is produced by a predecessor. In this case, weighted round-robin scheduling is a reasonable approach, since a task and its successors can execute concurrently in a pipelined fashion.

2.3.3 Priority Driven Scheduling Approach

The scheduling algorithms that fall under this approach are based on priority assignment to one or more of the temporal parameters of the tasks. These priority driven scheduling algorithms are divided into two categories: fixed-priority and dynamic priority algorithms. This section introduces four basic and highly used scheduling algorithms that fall into one of the above categories.

Fixed Priority Scheduling (FPS): The priority of the tasks being scheduled remains constant. The schedule is first made and then is implemented on the processors. Following two algorithms use fixed priority scheduling:

1. Rate Monotonic Scheduling
2. Deadline Monotonic (Inverse Deadline) Scheduling

Dynamic Priority Scheduling (DPS): The priority of the tasks being scheduled changes with the change in the parameter in consideration. The schedule changes dynamically as the priorities of the tasks change. The following algorithms are based on dynamic priority:

1. Earliest Deadline First

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
2. Least Slack Time First

These algorithms are further explained in detail in Chapter 3.

2.4 Scheduling Policies in Real Time Systems: Classification

Scheduling algorithms can be classified according to the following criteria:

- Off-line/On-line scheduling
- Preemptive/Non-preemptive scheduling
- Centralized/ Distributed scheduling

Figure 2 shows a possible classification of algorithms.

![Fig 2 Classification of RTES Scheduling Algorithms](image)

2.4.1 Off-line/On-line (Static/ Dynamic)

Off-line Scheduling (Static Scheduling): A scheduling algorithm is used offline if it is executed on the entire task set before actual task activation. The schedule generated in this way is stored in a table and later executed on the processors. The task set has to be
fixed and known a priori, so that all task activations can be calculated off-line. The main advantage is that the run-time overhead is low and it does not depend on the complexity of the scheduling algorithms used to build the schedule. However, the system is quite inflexible to environmental changes. Environmental changes here refer to the changes in the environment of operation of the system and processors like power supply and voltage fluctuation which lead to change in the task parameters.

On-line Scheduling (Dynamic Scheduling): A scheduling algorithm is used on-line if scheduling decisions are taken at run-time every time a new task enters the system or when a running task terminates. With on-line scheduling algorithms, each task is assigned a priority, according to one of its temporal parameters. These priorities can be either fixed priorities, based on fixed parameters and assigned to the tasks before their activation, or dynamic priorities, based on dynamic parameters like voltage, frequency of operation and execution time that may change during system evolution. This dynamic approach provides less precise information for scheduling the tasks than the static approach since it uses less information, and it has higher implementation overhead. However, it manages the unpredictable arrival of tasks and allows progressive creation of the planning sequence. Thus, on-line scheduling is used to cope with aperiodic tasks and abnormal overloading.

2.4.2 Preemptive/Non-preemptive

Preemptive Scheduling: In preemptive scheduling, an elected task may be preempted and the processor allocated to a more urgent task or one with higher priority; the preempted task is moved to the steady state, awaiting later election on some processor. Preemptive scheduling is usable only with preemptive tasks.
Non-preemptive Scheduling: Non-preemptive scheduling does not stop task execution. One of the drawbacks of non-preemptive scheduling is that it may result in timing faults that a preemptive algorithm can easily avoid. In uniprocessor architecture, critical resource sharing is easier with non-preemptive scheduling since it does not require any concurrent access mechanism for mutual exclusion and task queuing. However, this simplification is not valid in multiprocessor architecture.

2.4.3 Centralized/Distributed

Centralized: Scheduling is centralized when it is implemented on a centralized architecture that records the parameters of all the tasks of a distributed architecture.

Distributed: Scheduling is distributed when each site defines a local scheduling policy. Distributed scheduling can be of various types. For example, every processor executing a set of tasks can have its own scheduling policy or a group of processors can have same scheduling policy and the others can have their own scheduling policy. In this context some tasks may be assigned to a site and migrate later.
CHAPTER 3

REVIEW OF EXISTING LITERATURE

3.1 Scheduling Algorithms: The Big Picture

With the vast amount of literature available on scheduling of real time systems, it is necessary for the reader to understand the hierarchy of the existing scheduling algorithms. This section captures the literature available on scheduling algorithms focused on minimizing the power and energy consumption in RTES. Our aim is to include all possible variations in the existing algorithms.

![Classification of Scheduling Algorithms into Fixed and Dynamic Priority Algos.]

The algorithms are classified by the priority they use in scheduling the tasks. The algorithms are classified as belonging to fixed priority category or to dynamic priority.
category. And they are further classified as belonging to one of the scheduling algorithms. And those algorithms which do not use any of the above scheduling algorithms but implement a different priority scheduling are discussed separately. The algorithms are also categorized according to the objective function they minimize. Figure 3 shows the classification picture. Before studying the literature, it is necessary to understand the task model and the concept of power and energy minimization.

### 3.2 Real Time Task Model

A task model is required as a basis for discussing scheduling. A real time task is a basic executable entity which can be scheduled; it can be either periodic or aperiodic with soft or hard timing constraint. A task is best defined with its main timing parameters. This model includes the following primary parameters

- **r** – Release Time of the task
- **e_t** – Worst Case Execution Time of the task
- **D_t** – Relative Deadline of the task
- **d_t** – Absolute Deadline of the task
- **p_t** – Period of the task (Valid only for periodic tasks)
- **g** – Start time of the task
- **h** – End time of the task

### 3.3 Voltage Scaling and Energy Minimization

Power and energy are the foremost objective functions that scientists around the world are trying to optimize while developing scheduling policies for real time systems.
The basic concept of power reduction in the variable voltage processors is a technique called Voltage-Clock scaling in CMOS circuit technology.

Assuming that a processor P runs at a supply voltage V, and a frequency \( f \), and a task T takes \( n \) clock cycles on the processor to complete its execution, the power consumption in the CMOS digital circuit is given by

\[
P_{\text{cmos}} = C_L N_{\text{sw}} V^2 f
\]

where \( C_L \) is the output capacitance, \( N_{\text{sw}} \) is the number of switches per clock.

Energy consumption can be computed as

\[
E = (n \cdot P_{\text{cmos}}) / f
\]

This can be written as

\[
E = n \cdot C_L N_{\text{sw}} V^2
\]

From the above equations, we can conclude that lowering the supply voltage drastically reduces the power and energy consumption of the particular processor.

The supply voltage also affects the circuit delay. The circuit delay is given by

\[
Td = k \cdot V/ (V-V_t) 2
\]

where \( k \) is a constant dependent on output capacitance, \( V_t \) is the threshold voltage.

Frequency of operation is inversely proportional to the delay. Hence, lowering the supply voltage increases the delay of operation, which in turn leads to lower clock frequencies.

Processors supporting several supply voltages are available. Various supply voltages result in different energy consumption levels for a given task T. The voltage at which a task is run can be decided before the execution of the task or sometimes when the task is executing. In the former method, a particular voltage is assigned to a task and it is run at
that voltage on the processor. In the latter case, the switching of the voltage while the tasks are running on the processor is controlled by the OS.

These variable voltage processors operate at different voltage ranges to achieve different levels of energy efficiency. Some processors which can operate at different supply voltages are:

- ARM7D – runs at 33MHz, 5V and 20MHz, 3.3V;[11]
- Motorola’s PowerPC860 – can be operated at 50MHz, 3.3V and in a low power mode of 25MHz and 2.4V; [11]
- PowerPC603 – Has four power modes which can be selected by setting the appropriate control bits; [6]

Fig 4 Standard Task Graph
3.4 Standard Task Graph (STG)

An STG file consists of the task graph part and information part.

Task graph part:

Line#1 represents the number of tasks. Line#2 holds the information for the dummy source node. It is represented as task number 0. All subsequent lines hold information about rest of the nodes in the graph. The first column represents the task number, second column gives its execution time, third column gives the number of predecessors and the rest of the columns represent the node number of the predecessors. Line#13 holds information of the dummy sink node.

Information part:

The information part consists of other information about the graph and the program that generates the STG. This part is composed of four different parts: a common part (task graph file name, etc.), precedence constraints form, task processing time, and other information such as critical path length and task graph parallelism. The common information part is shown in the line #15. Likewise the other major parts are also described in detail. Usually they serve the documentation purposes. Each line in the information part starts with a ‘#’.

3.5 Description of the Existing Scheduling Algorithms

3.5.1 Fixed Priority Algorithms

The scheduling algorithm is said to be fixed priority algorithm if the parameter based on which priority is assigned is fixed.

3.5.1.1 Rate Monotonic Scheduling [5]
A set of tasks are said to be scheduled based on Rate Monotonic Scheduling if the priorities to the tasks are assigned according to their periods: the shorter the periods, the higher the priority of the task.

Liu and Layland proposed Rate Monotonic Scheduling approach with the following assumptions:

- Tasks are periodic with constant interval between two successive requests.
- Task deadlines are equal to their periods i.e. the tasks must be completed before their next instance.
- Tasks are independent in the sense that they do not depend on completion of other tasks.
- Execution time of each task is constant and does not vary with time.
- Tasks are preemptable and are run on one processor.

This algorithm is best explained by the following example: A set of three tasks is given: T₁ (4,1), T₂(5,2), T₃(20,5). All three tasks are scheduled on one processor P₁. The first parameter is the period and next one is the execution time of each task. The priority ordering of the tasks is T₁, T₂, T₃ since p₁ < p₂ < p₃. The RMS schedule for these tasks is shown below:

Fig 5 Rate Monotonic Schedule of tasks T₁, T₂, T₃

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In this paper, *Low Power Fixed Priority Scheduling* (LPFPS), a power efficient version of the fixed priority scheduling algorithm is proposed. Tasks are scheduled by fixed priority RMS algorithm. Power reduction is obtained by exploiting the slack times present in the system and those arising from the variations of execution times of the tasks while ensuring that all tasks meet their deadlines. These slack times are efficiently used to change the frequency and voltage of a processor. A heuristic methodology is presented to compute the ratio of the processor's speed. The tasks are assumed to be periodic, independent and are preemptable. The analysis of the algorithm was done as given below:

**Microprocessor Core**

ARM8 - Max clock frequency: 100MHz; Supply Voltage: 3.3V

**Benchmark Applications:**

- Avionics
- INS
- Flight Control
- CNC

Among all the benchmark applications, LPFPS obtains the most power gain of up to 62% for INS.

3.5.1.1.2 Power optimization of Real Time Embedded Systems on variable speed processors [7]

The authors of this paper proposed a power optimization method for real-time embedded applications on a Variable Speed Processor (VSP) with a *power-down mode*. This method consists of two components: *off-line component based on real time analysis of a task set* and an *on-line component based on priority based real time scheduling*.
(RMS). Specifically for a given real-time task set, the lowest possible maximum processor speed is computed such that at least one of the deadlines are violated if the processor is running below that speed. With the maximum speed of the VSP set to the computed value, the speed of the VSP is then dynamically varied or the VSP is brought into a power-down mode to exploit execution time variation of each task and idle intervals present in the schedule. The analysis is performed with similar specifications as that of the [6] and CNC application benchmark shows a maximum of 50% power reduction with each method compared to the conventional priority based scheduling.

3.5.1.1.3 Energy efficient Fixed-priority Scheduling of Real-Time Systems on Variable Voltage Processors [8]

The problem of determining the optimal voltage schedule for a real time system with fixed priority jobs implemented on a variable voltage processor is discussed in this paper. This technique is based on the assumption that the timing parameters if each job is known off-line. Two algorithms are presented in the paper. The first one takes $O(N^2)$ time ($N$ is the number of jobs) to find the minimum constant speed needed to complete each job, since constant voltage tends to result in a lower power consumption. The second algorithm, with $O(N^3)$ time complexity, builds on the first one and gives two results: (i) the minimum constant voltage needed to complete a set of jobs, and (ii) a voltage schedule which always results in lower energy consumption compared to using the minimum constant voltage and shutting down the system when it is idle.

The type of real time system implemented here consists of jobs with predefined release times, deadlines and required number of CPU cycles. These jobs can be aperiodic or be instances of periodic tasks and are scheduled by a preemptive scheduler following...
RMS policy. The performance of this approach has been compared with [6] and [7] and the results show that [8] has 60% more energy savings than [6] and almost equivalent and sometimes more energy savings than [7].

3.5.1.1.4 Scheduling and Assignment for Real-time Embedded Systems with Resource Contention [9]

This paper attempts to minimize the schedulability loss due to blocking. Blocking occurs when a priority inversion makes a higher priority task waiting for the processing of a lower priority task. The authors propose a method which deals with resource constraint for heterogeneous hard real-time systems and which tries to minimize the blocking time.

The scheduling and the assignment problems are interdependent. Hence the complexity is reduced by first assigning for each task, the sub tasks on each Processing Unit (PU) in order to bind resources on Pus and to minimize the schedulability loss due to blocking. Then Priority Ceiling Protocol (PCP) along with RMS is performed at the task level in order to reduce the blocking time and avoid deadlock. This algorithm reduces the schedulability loss of an average of 12%. For more than 10%, the algorithm works for cases which didn’t be dealt with other methods.

3.5.1.2 Deadline Monotonic (Inverse Deadline) Scheduling [10]

Tasks with the shortest relative deadline get the highest priority. This algorithm was first proposed by Leung and Merrill in 1980. This algorithm is valid even when the relative deadline is less than the task period .When the relative deadline is equal to its period, the Rate Monotonic and Deadline Monotonic algorithms behave in the same manner. The following schedule illustrates the DMS algorithm: A task is represented as T
\( \{r_i, C_i, D_i, p_i\} \). The system consists of three tasks \( T_1 \{0, 3, 7, 20\} \), \( T_2 \{0, 2, 4, 5\} \), \( T_3 \{0, 2, 9, 10\} \).

![Deadline Monotonic Schedule of Tasks T1, T2, T3 in 8.1.2](image_url)

The DM algorithm can sometimes produce a feasible schedule when RM algorithm fails but RM algorithm always fails when DM fails.

### 3.5.2 Dynamic Priority Algorithms

The scheduling algorithm is said to be dynamic priority algorithm if the priorities are assigned to tasks based on parameters that change during task execution.

#### 3.5.2.1 Earliest Deadline First (EDF) [5]:

This dynamic scheduling algorithm was first proposed by Liu and Layland along with RMS policy. The same assumptions made for RMS also stand for EDF. Priorities are assigned to tasks according to their absolute deadline. The task with earliest deadline has highest priority. This algorithm is important because it is optimal when used to schedule tasks on a processor as long as preemption is allowed and tasks do not contend for resources and this algorithm is optimal in the sense of feasibility: if there exists a feasible schedule for a task set, then the EDF algorithm is able to find it.

Figure 7 shows an example of EDF schedule for a set of three periodic tasks.
$T_1 \{0, 3, 7, 20\}, T_2 \{0, 2, 4, 5\}$ and $T_3 \{0, 1, 8, 10\}$.

![Fig 7 Earliest Deadline First Schedule of Tasks T1, T2, T3 in 8.2.1](image)


The authors here make several assumptions to apply clock scaling with EDF scheduling –

- Voltage switching consumes negligible overhead.
- Tasks are independent.
- The worst case execution time of each task is known.
- The overhead of the scheduling algorithm is negligible when compared to the execution time of the application.
- The system operates at two different voltage levels.

This algorithm is implemented in two phases: (i) Mode assignment (ii) Resource reclaiming phase. All Mode assignment picks the voltage setting for each task, i.e. High or Low voltage mode that will minimize the total energy consumed and ensure schedulability under EDF. This voltage assignment is done based on the amount of slack period left by the previously run task on the system. The slack available is computed after
the completion of execution of a task and the assignment is done in the order of tasks’ first arrival instances during each busy cycle. The slack if available is kept in a different queue which is checked every time a new task instance is released to see if there is any slack that can be reclaimed. The results have been published for three types of voltage assignments: fixed, static and dynamic. For dynamic mode assignments, the subsets of high and low voltages are determined for each busy cycle during which the processor is busy continuously without any idle intervals, while the static mode assignment assigns the voltage at the start of the task instance as the task timings are known a priori. In addition to the above two schemes, fixed voltage assignment follows the static assignment but with no slack reclamation. In the extreme case, more than 25% of the energy can be saved by the dynamic approach over the fixed assignment.

3.5.2.1.2 Real-Time Task Scheduling for a Variable Voltage Processor [12]

This work addresses the problem of less energy consumption by simultaneously assigning the CPU time and a supply voltage to each task which results in low power. This algorithm uses a Variable Voltage Processor core to schedule the tasks. Three methodologies are presented for voltage scheduling of the tasks. (i) Static Voltage scheduling of the tasks first assigns CPU time to all the tasks based on EDF schedule and then a supply voltage is assigned to each task to minimize the total energy consumption without violating the real time constraints. (ii) Dynamic voltage scheduling assigns a supply voltage to only one task which is going to run next. An occupation period is defined as the maximum value of period that the next executed task can use without violation of real time constraints for the future tasks. Two algorithms are proposed to calculate the occupation period: SD and DD algorithms. The start point is determined as
the actual finishing time of the current task or arrival of the next task. In DD algorithm, the end point is determined as the finishing time of the next task on the assumption that all tasks are assigned the maximum supply voltage and complete at the worst case execution cycle. In the SD algorithm, the end point is determined as the time which is added the minimum remaining time in not yet executed tasks to the end point in the DD algorithm. These are two optimal algorithms which maximize the length of the occupation period. These algorithms have been tested on a set of tasks which are run on a processor that can operate in three different modes.

The modes of the processor are 5v, 50MHz; 4v, 40MHz; and 2.5v, 25MHz. The five tasks $T_1$, $T_2$, $T_3$, $T_4$, $T_5$ are described with certain execution cycles and load capacitances. Two scenarios of task execution are assumed where in the deadlines of the tasks in scenario 1 are earlier than in scenario 2. For scenario 1, energy reduction rate of SS, SD and DD are 27%, 38% and 16% respectively compared with normal. In scenario 2, the energy reduction rate of SS, SD and DD are 56%, 58% and 16% respectively.

3.5.2.1.3 Energy Aware EDF Scheduling in Distributed Hard Real Time Systems [13]

An online energy aware algorithm for distributed heterogeneous hard real time systems based on a modification of the Earliest Deadline First algorithm. A distributed hard real time system that consists of a set of independent periodic tasks where each task is specified by its period, worst case execution time and deadline. An important assumption made here is that all tasks are statically allocated to a specific host during the system design phase. Low Power Distributed EDF (LPDEDF) is the energy aware algorithm. The energy reduction is obtained by reducing the speed of the processor when the only ready task in the system has no successor or when the previous task has not
exhausted its WCET. For performance analysis, the number of hosts has been randomly chosen between 2 and 4. The number of end to end deadline have been chosen as $2^*$ number of hosts. The number of tasks has been chosen between number of hosts and $6^*$ number of hosts. Task periods have been randomly chosen between 100 and 1000. The load of each task is determined by dividing randomly the fixed total load of the system. The end to end deadlines are equal to their periods. The proposed algorithm reduces energy consumption from 10% to 16% where the global load of the system is 65%. In the most loaded case of 95%, the energy savings are between 2% to 10%.

3.5.2.1.4 Energy Conserving Feedback EDF Scheduling for Embedded Systems with Real-Time Constraints [14]

This work attempts to enhance the EDF scheduling to exploit slack time generated by the invocation of the task at multiple frequency levels within the same invocation. Initially the overall system utilization is determined to configure the idle task. At each scheduling point for task activation, the scaling level is calculated and the scaled task portion is scheduled. If a task was preempted, the newly released task receives its recalculated slack prior to scaling. Slots from idle jobs increase the slack. In the absence of slack, there is no scaled portion and the task proceeds to execute at the highest frequency. Otherwise, a timer interrupt is set at the end of the scaled portion. The resulting energy savings exceed those of previously published work by up to 34%.

3.5.2.1.5 How to Integrate Precedence Constraints and Shared Resources in Real-Time Scheduling [15]: Previously the Priority Ceiling Protocol (PCP) and the Stack Resource Policy (SRP) have been used without precedence constraints. In this paper, Dr. Stankovic and Dr. Spuri extended these protocols to work with arbitrarily timed tasks with
precedence constraints. They characterize the EDF scheduling policy to correctly schedule precedence constrained tasks and show how preemptive algorithms, those that deal with the shared resources can be extended to deal with precedence constraints. They prove these mathematically using a set of theorems.

3.5.2.1.6 Characteristics of EDF Schedulability on Uniform Multiprocessor [16]

EDF algorithm has been traditionally used to schedule the real time systems on a single processor system. The authors of this paper develop tests for determining whether the EDF algorithm can successfully schedule a given real-time task system to meet all deadlines upon a specified uniform multiprocessor system. They attempt to efficiently identify all those uniform multiprocessor platforms such that any real-time instance feasible upon these platforms is guaranteed to be EDF schedulable upon the platform under consideration. EDF schedulability upon the given platform is then determined by ascertaining whether the real time system is feasible upon any of these platforms.

3.5.2.1.7 A Scheduling Model for Reduced CPU Energy [17]

As seen from the title of the paper, the authors here present a novel scheduling model for minimizing the energy consumed by the CPU. One very important assumption the authors make is that the energy consumption of a schedule is a convex function of the processor speed. They present an offline scheduling methodology to minimize the energy function. In the offline scheduling algorithm, a critical interval is defined which is an interval in which a set of jobs must be scheduled at maximum, constant speed in any optimal schedule. The algorithm proceeds by identifying such critical intervals, scheduling those critical jobs by following the EDF policy, then constructing a sub problem for the remaining jobs and solving it recursively. In the online scheduling
category, the authors identify two heuristics: (i) Average Rate Heuristic (ii) Optimal Available Heuristic. The Average Rate Heuristic associates with each job an average rate requirement or density. At any time t, the AVR sets the processor speed and uses the EDF policy to choose among the available jobs. In the Optimal Available schedule, a new optimal schedule for the problem instance is recalculated after each arrival of a new job and the remaining portions of all other available jobs.

3.5.2.1.8 Online Scheduling of Hard Real-Time Tasks on Variable Voltage Processor

This paper concentrates on scheduling a mixed set of tasks containing periodic tasks as well as sporadic tasks to optimize power consumption. The authors first discuss the scheduling of sporadic tasks and then they discuss the scheduling of mixed task sets. When only sporadic tasks arrive in the system, first an acceptance test using the processor utilization as the parameter is performed to check whether the task can be scheduled. The task is accepted only if it passes the acceptance test. All the tasks are maintained by EDF priority. These tasks are then scheduled on a variable voltage processor by using Sporadic Task Scheduling (STS) algorithm. For scheduling the set of mixed tasks, the authors came up with two algorithms: (i) OPASTS – Optimal Periodic And Sporadic Task Scheduling (ii) HPASTS – Heuristic Periodic And Sporadic Task Scheduling. The first is optimal when there is no knowledge of the arrivals of sporadic tasks. It has a time complexity of O (N+m) where N is the total number of requests in each hyper period of the n periodic tasks in the system and m is the number of sporadic tasks that have been accepted. The time complexity of the HPASTS is O (m). This algorithm is not optimal, but efficient and effective. The proposed efficient algorithms result in scheduling
solutions for various scheduling scenarios and workloads, which are within 20% of the minimum bound achievable with the dynamically variable voltage approach.

3.5.2.1.9 Real Time Task Scheduling for Energy-Aware Embedded Systems [19]

The authors of this paper propose two methodologies to schedule periodic, non-preemptable tasks in a real time system to minimize the energy consumption. To develop these policies, they assume that they are given a set of \( n \) periodic tasks with each task having a release time, deadline, execution length and period. They assume that the CPU can operate at two voltage levels and the supply voltage is controlled by the OS. The first scheduling model is (i) MILP – Mixed Integer Linear Programming Model: Here the objective function of ILP is the energy consumed by the set of \( n \) tasks with the following constraints –

- CPU speed is limited to two values \( s_1 \) and \( s_2 \).
- The deadline for each task must be met.
- Tasks are non preemptable
- A task may start only after it has been released.
- This model is computationally intensive and cannot be used for large task sets.

The second scheduling policy is (ii) LEDF – Low Energy Earliest Deadline First Heuristic. This algorithm maintains a list of all released tasks based on EDF scheduling policy. When tasks are released, the task with the earliest deadline is chosen first to be executed. Initially, a check is performed to see whether the task meets its deadlines when it is executed at a lower speed. If the task passes the test, the task is assigned the lower voltage and it begins execution. Any task that enters the system during this period is put in the ready list. LEDF recursively selects the task with the earliest deadline for
execution. As long as there are tasks for execution, the processor is not idle and this procedure is repeated until all the tasks are scheduled. The LEDF algorithm can be used for large real time task sets. As per the results, LDEF and MILP fare the same till the number of tasks are 14, after which the LEDF consumes slightly higher power than MILP. This is because LEDF does not know the release times of the tasks in a priori. But the time of generating an optimal schedule is less for LEDF when compared to MILP while scheduling a large set of tasks.

3.5.2.2 Least Slack Time First (LST) [20, 21]

This LST algorithm first proposed by Dhall and Sorenson assigns priority to tasks according to their slack (laxity): the smaller the slack, the higher the priority. If the slack is calculated only at the arrival times, the LST is equivalent to the EDF schedule.

Fig 8 Least Slack Time First Scheduling of Tasks T1, T2, T3 in 8.2.2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER 4

DESCRIPTION OF THE SCHEDULING ALGORITHMS

4.1 Mathematical Modeling of the Scheduling Problem

For the problem of scheduling, we consider a task set $T = \{T_1, T_2, T_3 \ldots T_n\}$ where each task $T_i$ is an aperiodic task with a hard deadline $d_i$ and Worst Case Execution Time (WCET) of $e_i$. These tasks execute on a multiprocessor system consisting of processing elements $\{PE_1, PE_2, PE_3, \ldots PE_m\}$ which can operate at a prespecified set of voltages $V = \{V_1, V_2, V_3 \ldots V_p\}$ where $i < j$ implies $V_i > V_j$. We denote a task $T_i$ running at voltage $V_k$ as $T_k$. The WCET of a task $T_i$ is its execution time at maximum voltage $V_1$ denoted by $e_{i1}$. Thus, its execution time at voltage $V_k$ is given by

$$e_{ik} = e_{i1}(V_1/V_k)$$

$T$ is subject to precedence constraints and a partial order $t_i < t_k$ is defined on $T$ where $t_i < t_j$ implies $t_i$ precedes $t_j$. The problem of scheduling may be mathematically defined as a mapping

$$\sigma : T \rightarrow \{PE_0, PE_1, \ldots , PE_m\}$$
$$\delta : PE \rightarrow \{V_1, V_2, \ldots , V_p\}$$
$$\tau : T \rightarrow t$$

where $t$ denotes the time axis, $0 \leq t \leq D$

such that $\forall T_i \in T, \tau(T_i) < d_i$

$D$ is the absolute deadline of the last task.
4.2 Algorithm Description

A real time system is modeled as a task graph where each task is represented by a node and the dependencies between the nodes are represented by edges between the nodes. Henceforth in this thesis, the term 'task' and 'node' mean the same.

4.2.1 The Scheduling after Scaling (SaS) Algorithm

This algorithm proceeds by scaling the voltage at which each task should execute and progressively scheduling the task on to the available processor. The Standard Task Graph (STG) input benchmarks used during the algorithm development provide only the execution time and the predecessors of a task. Since the tasks in consideration have arbitrary hard deadlines, we first proceed by determining the deadlines of all the tasks. The following steps illustrate how the algorithm proceeds –

Step 1: Finding the path with highest execution time

The first step is to find the path from the start node to the sink node in the task graph with highest execution time. This is achieved by exhaustive depth first search. The highest execution time is termed as $E_g$.

Step 2: Finding the deadlines of the tasks by backtracking

The deadlines of all the tasks are determined once the highest execution time is found. This is accomplished by backtracking to each node from the sink node. The deadline of the last node is found by multiplying $E_g$ with an arbitrary factor ‘x’. ‘x’ is selected such that the tasks have sufficient slack allowing them to be scaled.

$$d_{(sink\ node)} = E_g \times x$$

The deadlines of all other tasks are found by backtracking from the last node. The procedure of finding the deadlines is represented by the following equations.
\[ d_{i-1} = d_i - e_i \]

If a task has more than one successor, then the deadline is obtained by choosing the minimum of the differences of the deadlines and execution times of the successors. Suppose that a task \( i-1 \) has successors \( i, i+1, i+2 \), then \( d_{i-1} \) is given by

\[ d_{i-1} = \min \{ (d_i - e_i), (d_{i+1} - e_{i+1}), (d_{i+2} - e_{i+2}) \} \]

Slack of each node is found from the difference of deadline and execution time of each task.

slack time of each node \( s_i = d_i - e_i \);

Step 3: Scaling the voltage of each task

Once all the parameters of a task are determined, the next step is to scale the voltage of each task. It is assumed that the tasks can only execute at a prespecified set of voltages. These are the voltages at which the processors can operate.

First the tasks are sorted into set \( S \) with respect to their slack. Then the task with least slack is made the current node and its execution time is modified by a scaling factor. Then the start times and end times of all the successors of the current node are modified till the sink. If any node misses its deadline, then the scaling factor is reduced. Again, the start and end times of the successors are modified to see if any of the nodes misses its deadline. This procedure is repeated till none of the node misses it’s deadline and the voltage at which the current node executes is fixed. The scaling is performed on all the nodes in the graph. This scheme assures that all the nodes meet their deadlines and results in lowered voltages. This scheme is best explained by the pseudo code below:

Let the given set of voltages \( V = \{ V_1, V_2, V_3 \ldots V_p \} \) with \( V_1 \) being the highest and \( V_p \) being the lowest voltage at which a processing element can operate.
for (a task C in S)
{
    k = p;  //p is the number of voltage levels available for processors
    \[ e'_C = e_C \times \left( \frac{V_k}{V_{k'}} \right) \]
    Scale: modified execution time
    modified \( h_C' = g_C + e_{C\Box} \);
    if (\( h_C' < d_C \))
    for (all successor nodes N of C)
    {
        \( g_N = \max h' \) of all predecessors;
        \( h_N = g_N + e_N \);
        if (\( h_N > d_N \) and \( k > 1 \))
        {
            k = k-1;  //voltage level at which task executes increased
            goto Scale;
        }
    }
    else
    {
        k = k-1;  //increase voltage level
        goto Scale;
    }
}

\( V_k \) obtained at end of this code is the voltage at which the task C can run without any successor task missing its deadline. When run for the entire task set S, the mapping \( T \rightarrow \{V_1, V_2, ..., V_p\} \) is obtained.

Step 4: Scheduling the nodes on Unlimited Number of Processors
As step 3 ends, all the nodes are divided into voltage partitions with each node existing in exactly one partition. Scheduling is performed separately for each partition. By the end of the loop, \( V_k \) is the voltage to which the current node's execution time is scaled. At every instant of time, whenever start time of a node is encountered, a check is performed to see if any processor is idle. If any processor is idle, the node is scheduled on that processor, else a new processor is initialized. The pseudo code for scheduling the nodes on the processors is given below:

\[
\begin{align*}
&\text{Let } S \text{ be the set of tasks obtained by sorting all tasks with respect to their start times;} \\
&\text{Assign first task to first processor } PE_0 \\
&\text{Let } P \text{ denote the set of processors that have been allotted at least one task}
\end{align*}
\]

\[
\begin{align*}
&\text{for (every task } C \text{ in } S) \\
&\quad \begin{cases} \\
&\quad \text{if } (g_C > h_i \text{ of all tasks } T_i \text{ allotted to processor } P_j \in P) \\
&\quad \text{allot } C \text{ to } P_j \\
&\quad \text{else} \\
&\quad \text{allot } C \text{ to new processor } P_k \text{ where } k = |P| + 1 \\
&\end{cases}
\end{align*}
\]

This algorithm finally returns the number of processors required for each voltage partition. This results in minimum power and maximum resources.

4.2.2 Scheduling before Scaling Algorithm (SbS)

This algorithm performs scheduling of the nodes on the available processor before scaling the node voltages. It is similar to the SAS algorithm with the only difference
being that the nodes are initially sorted with respect to their deadlines following Earliest Deadline First policy. The nodes are then scheduled on the processors. At every instance of a start time of a node, a check is done to see if any processor is idle. The node is scheduled on the idle processor if any; else a new processor is initialized. Assuming that a processor can operate at only one voltage, all the nodes on a single processor are then scaled such that none of them miss their deadlines.

The pseudo code for the algorithm is given below:

Let \( S \) be the set of tasks obtained by sorting all tasks with respect to their deadline;

Assign first task to first processor \( PE_0 \)

Let \( P \) denote the set of processors that have been allotted atleast one task

for (every task \( C \) in \( S \))
{
    \hspace{1cm}
    \text{if} (g_C > h_i \text{ of all tasks } T_i \text{ allotted to processor } P_j \in P)$
    \hspace{1cm} allot \( C \) to \( P_j \)
    \hspace{1cm} else
    \hspace{1cm} allot \( C \) to new processor \( P_k \) where \( k = |P| + 1 \)
}

The scheduling till this stage corresponds to worst case scheduling and gives the minimum number of processors required to execute a task set. After scheduling, the voltages of the nodes are scaled. The pseudo code for scaling of the nodes on the processor is given below:

Let \( S \) be the set of processors on which the nodes are scheduled.

Sort the processors in ascending order based on the number of nodes on each
processor.

\[ S = \{ PE_0, PE_1, \ldots, PE_n \} \]

for (each processor PE)

for (a task C in PE)

\[
k = p; \quad \text{//p is the number of voltage levels available for processors}\]

Scale: modified execution time \( e'_c = e_c \times \left( \frac{V_1}{V_k} \right) \)

modified \( h'_c = g_c + e'_c \);

if \( (h'_c < d_c) \)

for (all successor nodes N of C)

\[
g'_N = \max h' \text{ of all predecessors};
\]

\[
h'_N = g'_N + e_N ;
\]

if \( (h'_N > d_N \text{ and } k > 1) \)

\[
k = k-1; \quad \text{//voltage level at which task executes increased}\]

goto Scale;

for (all nodes on processor PE executing after C)

\[
g_{c+1}' = h'_c ;
\]

\[
h_{c+1}' = g_{c+1}' + e_c ;
\]
if \((h_{C+1}' > d_{C+1})\)

\[ k = k-1; \quad //\text{voltage level at which task executes increased} \]

goto Scale;

\}

for (all successor nodes \(N\) of nodes on processor \(PE\))

\{

\[ g'_N = \max h' \text{ of all predecessors}; \]

\[ h'_N = g'_N + e_N; \]

if \((h'_N > d_N \text{ and } k > 1)\)

\[ k = k-1; \quad //\text{voltage level at which task executes increased} \]

goto Scale;

\}

else

\[ k = k-1; \quad //\text{increase voltage level} \]

goto Scale;

\}

\}

4.2.3 Probability Based Scheduling (PbS) Algorithm

The previous two algorithms have shown how to reduce the voltage of execution of each node the processors there by reducing the energy consumed. The next algorithm known as ‘Exhaustive Probability Based Scheduling’ tries to increase the utilization of the processors while optimizing the resources and the energy consumed by the system by including voltage scaling.
Step 1: Determining the time steps

All the tasks in the task set are assumed to be executing at their WCET i.e. all the tasks are currently running at 5V. Divide the final dead line of the graph into time-steps with the smallest execution time of all tasks as the unit of time steps.

Step 2: As Soon As Possible Scheduling

The next step in the algorithm is to perform As Soon As Possible Algorithm (ASAP). The ASAP Algorithm starts with the highest nodes (that have no parents) in the task graph and assigns time steps in increasing order as it proceeds downwards. It follows the simple rule that a successor node can execute only after its parent has executed. This algorithm clearly gives the fastest schedule possible. In other words, it schedules in least number of time steps but never takes into account the resource constraints.

Step 3: As Late As Possible (ALAP) Scheduling

The next step to follow in the algorithm is the As Late As Possible scheduling. The ALAP algorithm works exactly in the same way as the ASAP algorithm expect that it starts at the bottom of the task graph and proceeds upwards. This algorithm gives the slowest possible schedule that takes the maximum number of time steps. However this doesn't necessarily reduce the number of functional units used.

Step 4: Determining the mobility

The algorithm proceeds by finding the mobility of each task. Mobility of a task $T_i$ is defined below:

$$Mobility \mu = \text{ALAP\_begintime}[ T_i ] - \text{ASAP\_begintime}[ T_i ];$$

$\mu$ represents the number of different time steps in which the current task can be scheduled.
Step 5: Determine the probability

The next step in the algorithm is to find the probability of occurrence of each task in each time step covered by the mobility of the task.

The probability of a task C in a time step t is represented as \( P(C_t) \).

\( y = \text{Total time steps which includes all possibilities of occurrences of the task in its mobility.} \)

\( x = \min \{ \text{execution time units } e, \mu + 1 \} \)

\( z = y - 2(x - 1); \) Here \( z \) represents the number of times \( x \) is repeated.

function probability_of_tasks( C, P(C_t))

\[
\begin{align*}
\mu + 1 &= ALAP\_begintime(T_i) - ASAP\_begintime(T_i) + 1; \\
k &= 1; \\
\text{for } (i=1; i<=x-1; i++) \\
\{ \\
\quad P(C_t = k) &= i(1/(\mu + 1)); \\
\quad k++; \\
\} \\
\text{for } (i=1; i<=z; i++) \\
\{ \\
\quad P(C_t = k) &= x(1/(\mu + 1)); \\
\quad k++; \\
\} \\
\text{for } (i=x-1; i>=1; i--) \\
\end{align*}
\]
```c
P(Ct=k) = x(1/(l+1));
k++;
```

Step 6: Distribution Graph

A distribution graph is the summation of probabilities of each task in each time step. It signifies the portion of a task that will be executed in a time step.

Distribution graph \( DG(t_i) \) for a time step \( t_i \) is given as

\[
DG(t_i) = \sum P(C_j t_i)
\]

Step 7: Schedule the task in a time step

At this point, the current task has to be scheduled starting at a particular time step and ending at another. This is necessary in order to determine the change in probability of the task in each time step when scheduled.

Change in probability of a task \( C \) is denoted by \( \delta (P(C_j) \).

if the current task is scheduled to execute in time step \( t_i \) then change in probability is

\[
\delta (P(C_j t_i)) = 1 - P(C_j t_i)
\]

if the current task is not scheduled to execute in time step \( t_i \), then change in probability is

\[
\delta (P(C_j t_i)) = 0 - P(C_j t_i)
\]

Step 8: Self Effect of the nodes

Here the algorithm tries to balance the distribution graph by calculating the effect of each task to time step assignment and then selects the smallest effect.
Self effect of task $C_j$ in a time step $t_i$ is

$$SE(C_j t_i) = DG(t_i) \cdot \delta(P(C_j t_i))$$

The total self effect for a task is given by

$$\sum SE(C_j t_i)$$

Step 9: Successor Effect

Scheduling the current task directly effects the scheduling of the successor tasks. Hence the successor effect is also considered while determining the total force of a task. The successor force is also found out in the same manner as self force.

Step 10: Total Effect

The total effect of a task determines the feasibility of scheduling the task in that particular time step. The least effect determines the step in which the task has to be executed. This effect signifies that the current task will use fewer resources if scheduled in a time step where the self effect is least and vice versa.

Total effect of a task $C$ is

$$F(C_j t_i) = SF(C_j t_i) + \text{Successor Effect}$$

After finding the total effect, the task is scheduled in the time step where the effect is least. Steps 7, 8, 9 and 10 are repeated for all the tasks in the task set and a final schedule is obtained as a result.

Step 11: Voltage Scaling

This step is a very important step in this algorithm as this incorporates voltage scaling into the schedule obtained so far.

$$V = \{V1, V2, V3...Vk\} \text{ are the pre specified set of voltages at which a task can execute.}$$
for (all tasks in the existing schedule)
{
    \[ e'_c = e_c \times \left( \frac{V_1}{V_k} \right) \]
    
    \( k = p; \) // \( p \) is the number of voltage levels available for processors
    modified \( hC' = gC + eC'; \)
    
    if \((hC' < dC)\)
    {
        repeat steps 5, 6, 7, 8, 9 and 10;
        
        In step 9, the successors are assumed to be operating at the highest voltage possible;
    }
    
    else
    {
        \( k = k-1; \)
        
        \( k = k-1; \)
    }
}

The pseudo code shown above performs the voltage scaling of each task and the total effect is calculated for different voltages. The least effect determines the voltage at which the task will be executed. After a task is fixed in a voltage, the corresponding DG is updated. This process is repeated for all the tasks and each one is scheduled for execution at a particular voltage and the DG is updated dynamically to get an optimized schedule.

Step 12: Number of processors
The final step in this algorithm is to find the number of processors required to execute the scheduled task set. By the end of scheduling, all the tasks are ready to be executed at a specified voltage. The pseudo code to find the number of processors is given below.

\[
\text{for (each voltage)} \\
\text{allocate node with earliest start time to one processor} \\
\text{for (all tasks running at that voltage)} \\
\text{if (gC > hi of all tasks Ti allotted to processor Pj ∈ P)} \\
\text{allocate C to Pj} \\
\text{else} \\
\text{allocate C to new processor Pk where k = |P| + 1} \\
\text{for (each voltage)}
\]

This algorithm finally returns a schedule in which the task voltages are scaled making sure none of the tasks miss their deadlines and these tasks are executed on processors in such a way that the utilization of the processors is high.

4.2.4 Energy - Probability Based Scheduling (E-PbS) Algorithm

The PbS algorithm discussed so far results in a schedule that has reduced concurrency of tasks in a timestep in order to minimize the number of resources required. But this does not involve any component that tries to reduce the energy. To achieve this, an algorithm Energy-Probability based Scheduling (E-PbS) has been developed in this thesis. This is a result of slight variation of PbS algorithm where in a factor of voltage is
included in calculating the total effect of a task. Steps 1 to 10 are followed as described in section 4.2.3.

Step 11: Energy factor

From step 10, the total effect of the task at the highest voltage among the prespecified set of voltages is determined. Voltage scaling is performed on all the tasks in the schedule to determine the total effect of the task at different voltages. The pseudo code for this is similar to that given in Step 11 of Section 4.2.3.

The total effects so determined are at different voltages. For example, effect $f_1$ is at voltage $V_1$, effect $f_2$ is at voltage $V_2$. The comparison of these effects would be difficult when they are at different voltages. To compare and find the least effect, these effects are divided by the voltage at which they operate.

$$F_1 \rightarrow \text{Total effect at voltage } V_1$$
$$F_2 \rightarrow \text{Total effect at voltage } V_2$$
$$F_1/V_1 \rightarrow \text{Total effect per volt at } V_1$$
$$F_2/V_2 \rightarrow \text{Total effect per volt at } V_2$$

Now $F_1/V_1$ and $F_2/V_2$ are comparable. Normalizing the fractions, we get the following:

$$F_1/(V_1/V_2) \rightarrow \text{Normalized effect at voltage } V_1$$
$$F_2 \rightarrow \text{Normalized effect at voltage } V_2$$

The least effect determines the voltage at which the task will be executed. After a task is fixed in a voltage, the corresponding DG is updated. This process is repeated for all the tasks and each one is scheduled for execution at a particular voltage and the DG is updated dynamically to get an optimized schedule.
Step 12: This is the final step in E-PbS algorithm and is similar to step 12 in PbS algorithm.

4.3 An Example illustrating the algorithms

The example here explains the methodology of the algorithm. Figure 9 is an example task graph consisting of ten nodes with edges between them depicting the dependencies between the tasks. The start and sink nodes have null execution time. This task graph is in the Standard Task Graph (STG) format. This is represented in the form of a table as in table 1. The deadline and slack of each node is calculated by backtracking as described in Chapter 2 in Section 4.2.1.

Fig 9 Standard Task Graph Example

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>Node#</th>
<th>Exe-Time</th>
<th>#of Predec</th>
<th>Predec.No</th>
<th>Start-Time</th>
<th>End-Time</th>
<th>Deadline</th>
<th>Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11.5</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>20.5</td>
<td>11.5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>13</td>
<td>34.5</td>
<td>21.5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>12</td>
<td>23.5</td>
<td>11.5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>26.5</td>
<td>20.5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>26.5</td>
<td>22.5</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>4, 5</td>
<td>6</td>
<td>9</td>
<td>29.5</td>
<td>20.5</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>32.5</td>
<td>20.5</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>23.5</td>
<td>15.5</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>2</td>
<td>3, 8</td>
<td>12</td>
<td>21</td>
<td>32.5</td>
<td>11.5</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>7, 9</td>
<td>21</td>
<td>23</td>
<td>34.5</td>
<td>11.5</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>2</td>
<td>2, 10</td>
<td>23</td>
<td>23</td>
<td>34.5</td>
<td>11.5</td>
</tr>
</tbody>
</table>

### 4.3.1 SaS

All the processors are assumed to be operating at one of the voltages in the voltage set \{5V; 3.3V; 2.4V\}. Eliminating the nodes with execution time of zero and applying the voltage scaling to the nodes sorted with respect to their slack, we get table 2. Node 1 has an initial execution time of 9. It is then multiplied by a factor of \(5/2.4\) and then the start and end times of the all successor nodes of 1, in this case \{2; 3; 9; 10\} are changed and checked to see if they meet their respective deadlines. In this case, none of the node misses its deadline; hence node 1 is allotted a voltage of 2.4. If any node misses its deadline, then we go back to node 1 and the scaling factor is changed to \(5/3.3\) and all the start and end times are changed to see if they meet their deadlines. If again, any node misses its deadline, the original node is set a default voltage of 5. Algorithm proceeds in...
this manner and is repeated for each node.

Table 2 Voltage scaling of each node

<table>
<thead>
<tr>
<th>Node#</th>
<th>Slack</th>
<th>ExeTime</th>
<th>Mod ExeTime</th>
<th>Mod StartTime</th>
<th>Mod EndTime</th>
<th>Comment</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.5</td>
<td>9</td>
<td>9(5/2.4) = 18.75</td>
<td>0</td>
<td>18.75</td>
<td></td>
<td>2.4</td>
</tr>
<tr>
<td>3</td>
<td>11.5</td>
<td>3</td>
<td>3(5/2.4) = 6.25</td>
<td>18.75</td>
<td>25</td>
<td>Misses deadline</td>
<td>3.3</td>
</tr>
<tr>
<td>9</td>
<td>11.5</td>
<td>9</td>
<td></td>
<td>23.29</td>
<td>32.29</td>
<td>Cannot scale further</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>11.5</td>
<td>2</td>
<td></td>
<td>32.29</td>
<td>34.29</td>
<td>Cannot scale further</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>15.5</td>
<td>8</td>
<td>8(5/2.4) = 16.67</td>
<td>0</td>
<td>16.67</td>
<td></td>
<td>2.4</td>
</tr>
<tr>
<td>4</td>
<td>20.5</td>
<td>6</td>
<td>6(5/2.4) = 12.5</td>
<td>0</td>
<td>12.5</td>
<td></td>
<td>2.4</td>
</tr>
<tr>
<td>6</td>
<td>20.5</td>
<td>3</td>
<td>3(5/2.4) = 6.25</td>
<td>12.5</td>
<td>18.75</td>
<td></td>
<td>2.4</td>
</tr>
<tr>
<td>7</td>
<td>20.5</td>
<td>3</td>
<td>3(5/2.4) = 6.25</td>
<td>18.75</td>
<td>25</td>
<td></td>
<td>2.4</td>
</tr>
<tr>
<td>2</td>
<td>21.5</td>
<td>4</td>
<td>4(5/2.4) = 8.33</td>
<td>18.75</td>
<td>27.08</td>
<td></td>
<td>2.4</td>
</tr>
<tr>
<td>5</td>
<td>22.5</td>
<td>4</td>
<td>4(5/2.4) = 8.33</td>
<td>0</td>
<td>8.33</td>
<td></td>
<td>2.4</td>
</tr>
</tbody>
</table>

Voltage scaling of all the nodes results in three partitions. In this case, the first partition operates at 5V, the next one operates at 3.3V and the last one operates at 2.4V. Scheduling algorithm is then applied to these partitions as shown in figure 10. By scaling the voltages, we make sure all the tasks meet their deadlines.

SaS scheme is applied to the example task set and the scheduling can be seen in figure 10. The total number of processors required with this scheme is 6. An assumption here is that each processor runs at a preset voltage. This leads to increase in the number of processors and also large idle time of the multiprocessor system. The total number of processors can be reduced by using processors that have voltage shift ability.
4.3.2 SbS

The SbS scheme is shown in Figure 11. The total number of processors required by this scheme is 4. The scaling here proceeds by scaling each node in each processor and checking to see if the current node or any of its successors or any of the nodes which are executing on the same processor as the current node do not miss their deadlines. One important thing to notice in this scaling is that all the nodes on one processor are scaled to same voltage because it is assumed that a processor can operate at only one voltage without having voltage shift ability.

In figure 11, first start by choosing processor $P_1$ and node1. Scale node1 by
multiplying it with a factor of \(\frac{5}{2.4}\) and check to see if it is meeting its deadline. And then modify the start and end times of all the successor nodes continuously making sure they meet their deadlines. Also the nodes on P1 that are executing after 1 i.e. 3, 9 and 10 also are modified for their start time and end time. If any node in this process misses its deadline, we trace back to the original node 1 and decrease the scaling factor to \(\frac{5}{3.3}\). This procedure is repeated for the new scaling factor and the node is fixed to that voltage. This procedure is repeated on all the nodes and the nodes in one processor are fixed to execute at only one voltage.

Fig 11 Scheduling before Scaling

Table 3 shows the tabular form of the procedure for scaling the nodes in SbS algorithm.
Table 3 Scaling in SbS algorithm

<table>
<thead>
<tr>
<th>Processor</th>
<th>Node</th>
<th>Exec time</th>
<th>Mod exec time</th>
<th>Mod start time</th>
<th>Mod end time</th>
<th>Comment</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>18.75</td>
<td>0</td>
<td>18.75</td>
<td></td>
<td>2.4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>6.25</td>
<td>18.75</td>
<td>25</td>
<td>Misses deadline</td>
<td>2.4</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>13.59</td>
<td>0</td>
<td>13.59</td>
<td></td>
<td>Nodes 2,3,9,10 meet their deadlines after modifying start and end times</td>
<td>3.3</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td></td>
<td>Nodes 2,3,9,10 meet their deadlines after modifying start and end times</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
<td>16.64</td>
<td>0</td>
<td>16.64</td>
<td>Nodes 9,10 and 7 meet their deadlines after modifying start and end times</td>
<td>2.4</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td></td>
<td>6.24</td>
<td>16.64</td>
<td>22.88</td>
<td>Node 10 meets its deadline</td>
<td>2.4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>6</td>
<td>12.48</td>
<td>0</td>
<td>12.48</td>
<td>Nodes 2,6,7,10 meet their deadlines after modifying the start and end times</td>
<td>2.4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td>6.24</td>
<td>12.48</td>
<td>18.72</td>
<td>Nodes 2,7,10 meet their deadlines after modifying the start and end times</td>
<td>2.4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td>8.32</td>
<td>18.72</td>
<td>27.04</td>
<td>Meets its deadline</td>
<td>2.4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td>8.32</td>
<td>0</td>
<td>8.32</td>
<td>All the successor nodes meet their deadlines</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Figure 12 Scaled processors in SbS algorithm

Processor P1 running at 5.0V

Processors P2, P3, P4 running at 2.4V

Figure 12 shows the final schedule obtained through SbS algorithm. As can be seen, processor P1 runs at 5V and all other processors are scaled to run at 2.4V.
4.3.3 Probability based Scheduling

The first two steps in this algorithm are to determine the ASAP and ALAP schedules of the task graph. Here the final deadline of the graph is 34.5 and hence, the whole graph is divided into 34 time steps with each time step equal to 1 time unit of execution. Figure 13 shows the ASAP and ALAP schedules for the example benchmark 10.stg. These schedules are required to calculate the mobility of each task. Mobility is calculated by the formula given in Step 4 of Section 4.2.3. Table 4 gives the mobility of each task along with the ASAP and ALAP start times of each task.

Table 4 Table showing ASAP and ALAP start times and Mobility of each task

<table>
<thead>
<tr>
<th>Node</th>
<th>Execution units</th>
<th>ALAP</th>
<th>ASAP</th>
<th>Mobility (μ+1)</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>9</td>
<td>12</td>
<td>1</td>
<td>12</td>
<td>20.5</td>
</tr>
<tr>
<td>[2]</td>
<td>4</td>
<td>31</td>
<td>10</td>
<td>22</td>
<td>34.5</td>
</tr>
<tr>
<td>[3]</td>
<td>3</td>
<td>21</td>
<td>10</td>
<td>12</td>
<td>23.5</td>
</tr>
<tr>
<td>[5]</td>
<td>4</td>
<td>23</td>
<td>1</td>
<td>23</td>
<td>26.5</td>
</tr>
<tr>
<td>[6]</td>
<td>3</td>
<td>27</td>
<td>7</td>
<td>21</td>
<td>29.5</td>
</tr>
<tr>
<td>[7]</td>
<td>3</td>
<td>30</td>
<td>10</td>
<td>21</td>
<td>32.5</td>
</tr>
<tr>
<td>[8]</td>
<td>8</td>
<td>16</td>
<td>1</td>
<td>16</td>
<td>23.5</td>
</tr>
<tr>
<td>[9]</td>
<td>9</td>
<td>24</td>
<td>13</td>
<td>12</td>
<td>32.5</td>
</tr>
<tr>
<td>[10]</td>
<td>2</td>
<td>33</td>
<td>22</td>
<td>12</td>
<td>34.5</td>
</tr>
</tbody>
</table>

The next step is to determine the probabilities of occurrence of each task in each time step covered by the mobility of the task. These probabilities are determined by using the formula given in Step 5 in Section 4.2.3. As an example, the probabilities for task1 are given in Table 5.
After determining the probabilities of each task in each control step, the next step is to find the distribution graph for each time step which is done by the summation of probabilities of all task occurrences in that time step. Table 6 gives one such DG for timestep1. Similarly, the DGs for each time step are determined.

Then the self force of the task is determined by scheduling the task in a control step and finding the variation in probability as given by the formula in steps 7&8 in section 4.2.3. The total self force is again the summation of all self forces of a task. Due to the high amount of data, the self forces of the tasks are not shown here. In the next step, the successor forces are calculated in a similar manner to the self force. The total force of a
task is then the addition of its self force and its successor forces. Through out this
procedure, the tasks are assumed to be running at 5V.

Table 5 Probabilities for task# 1 from ASAPbegin till ALAPend

<table>
<thead>
<tr>
<th>Time Step</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0833333</td>
</tr>
<tr>
<td>2</td>
<td>0.166667</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>0.333333</td>
</tr>
<tr>
<td>5</td>
<td>0.416667</td>
</tr>
<tr>
<td>6</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>0.583333</td>
</tr>
<tr>
<td>8</td>
<td>0.666667</td>
</tr>
<tr>
<td>9</td>
<td>0.75</td>
</tr>
<tr>
<td>10</td>
<td>0.75</td>
</tr>
<tr>
<td>11</td>
<td>0.75</td>
</tr>
<tr>
<td>12</td>
<td>0.75</td>
</tr>
<tr>
<td>13</td>
<td>0.666667</td>
</tr>
<tr>
<td>14</td>
<td>0.583333</td>
</tr>
<tr>
<td>15</td>
<td>0.5</td>
</tr>
<tr>
<td>16</td>
<td>0.416667</td>
</tr>
<tr>
<td>17</td>
<td>0.333333</td>
</tr>
<tr>
<td>18</td>
<td>0.25</td>
</tr>
<tr>
<td>19</td>
<td>0.166667</td>
</tr>
<tr>
<td>20</td>
<td>0.0833333</td>
</tr>
</tbody>
</table>

Table 6 Distribution Graph for Time Step-1

<table>
<thead>
<tr>
<th>Node#</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>0.0833333</td>
</tr>
<tr>
<td>[4]</td>
<td>0.047619</td>
</tr>
<tr>
<td>[5]</td>
<td>0.0434783</td>
</tr>
<tr>
<td>[8]</td>
<td>0.0625</td>
</tr>
<tr>
<td>Total DG in TimeStep 1</td>
<td>0.2369306</td>
</tr>
</tbody>
</table>

The next major step is to scale the voltages of the tasks in such a way that neither the
task nor its successors miss their respective deadlines. The schedule obtained so far is at
5V. Each task in this schedule is then scaled by a factor of (5/3.3) and (5/2.4) and the
total force for each task is determined for both the scaling factors. The least force among
the three forces at 5, 3.3 and 2.4V decides the force at which the task is to be executed.
While scaling a task, all the successors of the current task are assumed to be operating at
5V. After fixing a task at a certain voltage, the DG for each time step is updated and the
procedure is repeated for the next node available. The result of this dynamically updated
algorithm is an optimized schedule which uses a reduced number of resources. On the
other hand, the complexity of this algorithm is $3n^2$. The last and final step in this
algorithm is to determine the number of processors required to accommodate all the tasks
in the schedule. The processors here assumed to be operating at only one particular
voltage without the ability to shift voltages. This is accomplished as explained in Step12
of Section 4.2.3. The final schedule for the benchmark program 10.stg is shown in
Figure14.

4.3.4 E-PbS

The total force of a task at 5.0V, 3.3V and 2.4V are represented by $f_5$, $f_{3.3}$ and $f_{2.4}$
respectively. After determining the total forces, the next step is to normalize the forces
with the energy factor. The normalized forces are $f_5/(5.0/2.4)$, $f_{3.3}/(3.3/2.4)$ and $f_{2.4}$. The
least force among these normalized forces is selected and the task is scheduled to execute
at the corresponding voltage. The final schedule obtained by E-PbS is shown in figure 15.
Fig 14 Final PbS Schedule for 10.stg
Fig 15 Final E-PbS Schedule for 10.stg
CHAPTER 5

RESULTS, CONCLUSION AND FUTURE WORK

This work has focused primarily on generating optimal schedules for task sets representative of RTES on a multiprocessor architecture. Optimal schedules obtained in turn reduce the resources required by the system and the total energy consumed by the system. The work presented in this thesis is a small step in what we believe is the right direction, however there is obviously much more to be done. In the next section, a summary of what was done will be presented. This thesis concludes with future directions for RTES scheduling with communication costs between the tasks and with processors that have voltage shift ability.

5.1 Results

This chapter summarizes the results obtained by running the proposed algorithms on standard benchmarks. The four proposed algorithms produce four different schedules for a given scheduling problem. The main problem that has been addressed here is the number of processors available for scheduling a system. Initially, the number of processors is unknown and all four algorithms progressively determine the required number of processors. A comparison of the number of processors and their utilization in each of the algorithm is given in the later part of this chapter.
The other point of focus in this chapter has been the energy consumed by the system which is an important factor in the design and implementation of RTES. The energy consumed by the four algorithms while running the standard benchmarks is also reported in this chapter. The slack of each processor is also determined to find the efficiency of each algorithm. What follows is a brief discussion of the whole thesis followed by discussion about the input benchmarks, results and directions for future work.

5.2 Conclusion

Four different algorithms for scheduling and energy minimization of RTES by voltage scaling have been proposed in this thesis work. This thesis report starts with an introduction to RTES in chapter1. All relevant definitions and classification of existing scheduling techniques are given in chapter2. An extensive survey of scheduling algorithms proposed to reduce energy consumption in RTES is presented in chapter3. The four algorithms – Scheduling after Scaling (SaS), Scheduling before Scaling (SbS), Probability based Scheduling (PbS) and Energy-Probability based Scheduling algorithms are described in detail in steps followed by the explanation of an example benchmark in chapter4. A variation of PbS, Energy-Probability based Scheduling (E-PbS) is also described in chapter4. This helps the reader to understand the algorithms clearly. The pseudo code for part of the algorithms is provided wherever necessary. The current chapter forms the final chapter of this thesis with detailed results, conclusion and directions for future work provided in further sub sections.

5.2.1 Benchmarks

The Standard Task Graph (STG) template has been used as the input benchmark
through out the course of development of the scheduling algorithms.

5.3 Assumptions made

Following is a list of assumptions made during the course of development of the proposed SaS, SbS, PbS and E-PbS algorithms:

- All the processors operate at prespecified set of voltages (5.0V, 3.3V and 2.4V in this thesis.
- All the processors operate at a constant and same frequency.
- Processors used cannot shift from one voltage level to another while executing a task set.
- The switching capacitance of the processors is constant and same for all the processors.
- All the tasks are assumed to be aperiodic, dependent tasks.

5.4 Simulation Results

The simulation results obtained by running STG benchmarks are presented in this section. All four scheduling algorithms are completely implemented in C++ on Linux platform. The benchmarks used vary in size. The number of nodes vary from 7 to 500.

Table 7 shows the number of processors required by the system when the four algorithms are implemented separately.
Table 7 Table showing number of processors required

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Nodes</th>
<th>SsS 5.0V</th>
<th>SsS 3.3V</th>
<th>SsS 2.4V</th>
<th>Total</th>
<th>PSS 5.0V</th>
<th>PSS 3.3V</th>
<th>PSS 2.4V</th>
<th>Total</th>
<th>E-PSS 5.0V</th>
<th>E-PSS 3.3V</th>
<th>E-PSS 2.4V</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-stg</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>10-stg</td>
<td>10</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>11</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>50-stg</td>
<td>50</td>
<td>2</td>
<td>13</td>
<td>16</td>
<td>21</td>
<td>1</td>
<td>14</td>
<td>15</td>
<td>30</td>
<td>7</td>
<td>6</td>
<td>16</td>
<td>33</td>
</tr>
<tr>
<td>sparse-stg</td>
<td>96</td>
<td>1</td>
<td>62</td>
<td>64</td>
<td>129</td>
<td>1</td>
<td>61</td>
<td>62</td>
<td>124</td>
<td>12</td>
<td>44</td>
<td>59</td>
<td>14</td>
</tr>
<tr>
<td>100-stg</td>
<td>100</td>
<td>3</td>
<td>46</td>
<td>54</td>
<td>103</td>
<td>2</td>
<td>41</td>
<td>46</td>
<td>99</td>
<td>19</td>
<td>15</td>
<td>41</td>
<td>53</td>
</tr>
<tr>
<td>300-stg</td>
<td>300</td>
<td>1</td>
<td>0</td>
<td>120</td>
<td>121</td>
<td>1</td>
<td>0</td>
<td>122</td>
<td>123</td>
<td>49</td>
<td>15</td>
<td>57</td>
<td>121</td>
</tr>
<tr>
<td>500-stg</td>
<td>500</td>
<td>6</td>
<td>4</td>
<td>205</td>
<td>215</td>
<td>2</td>
<td>2</td>
<td>201</td>
<td>205</td>
<td>79</td>
<td>33</td>
<td>87</td>
<td>199</td>
</tr>
</tbody>
</table>

The first column gives the names of the STG benchmarks used for the purpose of comparison. The second column gives the number of nodes in each benchmark. In the STG benchmarks, as the number of nodes increase, the complexity of the graph increases. The table is divided into four main columns, each one representing one scheduling algorithm. Each of these main columns is then divided into four sub columns. The first of these subcolumns gives the number of processors running at 5.0V. The second sub column gives the number of processors running at 3.3V and the third sub column gives the number of processors running at 2.4V. Finally, the fourth column represents the total number of processors required by the system to execute the benchmark program.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 16 is a comparison of the total number of processors required to execute the standard benchmarks. Here, we can observe that all scheduling policies use same number of resources with PbS using the least among them. The number of processors required to execute the tasks increase linearly with the number of tasks.

Figure 17 Division of total number of processors for SaS Algorithm by Voltage
Figure 17 shows the visual diagram of number of processors running at 5, 3.3 and 2.4 voltage levels. As can be seen, the number of processors is high for 2.4V partition which tells us that more number of nodes have been scaled to execute at the least voltage. Figure 18 shows a similar trend for SbS algorithm.

Fig 18 Division of total number of processors for SbS Algorithm by Voltage

Fig 19 Division of total number of processors for PbS Algorithm by Voltage
The number of processors running at three different voltages executing the PbS algorithm and E-PbS are shown in figure 19 and figure 20 respectively. A very important observation that can be made is that the total number of processors are distributed evenly to work in all the voltage levels unlike the SaS and SbS algorithms where more number of processors were scheduled to run in 2.4 voltage level.

Table 8 shows the energy consumed by the multiprocessor system when the proposed algorithms are implemented. The unit of energy here depends on the units of execution time of the tasks in the graph. A typical unit for execution would be nanoseconds as in a RTES, in which case the unit for energy would be nanoJoules.
Table 8 Energy consumed by the processors while implementing the algorithms

<table>
<thead>
<tr>
<th>Bench mark</th>
<th>Nodes</th>
<th>5.0V</th>
<th>3.3V</th>
<th>2.4V</th>
<th>Total</th>
<th>5.0V</th>
<th>3.3V</th>
<th>2.4V</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.stg</td>
<td>7</td>
<td>125</td>
<td>181.17</td>
<td>204</td>
<td>510.17</td>
<td>450</td>
<td>0</td>
<td>192</td>
<td>642</td>
</tr>
<tr>
<td>10.stg</td>
<td>10</td>
<td>275</td>
<td>49.5</td>
<td>406.56</td>
<td>731.06</td>
<td>575</td>
<td>0</td>
<td>336</td>
<td>911</td>
</tr>
<tr>
<td>50.stg</td>
<td>50</td>
<td>975</td>
<td>372.9</td>
<td>2033.28</td>
<td>3381.18</td>
<td>1375</td>
<td>0</td>
<td>2484</td>
<td>3859</td>
</tr>
<tr>
<td>sparse.stg</td>
<td>96</td>
<td>1500</td>
<td>370.26</td>
<td>21438.7</td>
<td>23308.96</td>
<td>3050</td>
<td>0</td>
<td>21768</td>
<td>24818</td>
</tr>
<tr>
<td>100.stg</td>
<td>100</td>
<td>1000</td>
<td>326.7</td>
<td>5338.56</td>
<td>6665.26</td>
<td>1200</td>
<td>1044.78</td>
<td>5184</td>
<td>7428.78</td>
</tr>
<tr>
<td>300.stg</td>
<td>300</td>
<td>1325</td>
<td>0</td>
<td>31188.5</td>
<td>32513.5</td>
<td>2350</td>
<td>0</td>
<td>35520</td>
<td>37870</td>
</tr>
<tr>
<td>500.stg</td>
<td>500</td>
<td>3600</td>
<td>773.19</td>
<td>52069.4</td>
<td>56442.59</td>
<td>4150</td>
<td>2231.79</td>
<td>59220</td>
<td>65601.79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PbS 5.0V</th>
<th>3.3V</th>
<th>2.4V</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.stg</td>
<td>300</td>
<td>261.4</td>
<td>92.16</td>
</tr>
<tr>
<td>10.stg</td>
<td>650</td>
<td>141.6</td>
<td>213.12</td>
</tr>
<tr>
<td>50.stg</td>
<td>3900</td>
<td>762.3</td>
<td>1094.4</td>
</tr>
<tr>
<td>sparse.stg</td>
<td>16650</td>
<td>1416</td>
<td>14331</td>
</tr>
<tr>
<td>300.stg</td>
<td>6850</td>
<td>1906</td>
<td>2056.3</td>
</tr>
<tr>
<td>500.stg</td>
<td>41100</td>
<td>4421</td>
<td>14106</td>
</tr>
<tr>
<td>PbS 3.3V</td>
<td>E-PbS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.stg</td>
<td>500</td>
<td>71600</td>
<td>11151</td>
</tr>
</tbody>
</table>

Column 1 lists the names of the benchmark programs used to implement the algorithms. The second column in the table above gives the number of nodes in the benchmarks. Each algorithm is marked by a different color and each of them is divided into four columns. The first column in each algorithm represents the energy consumed by the processors running at 5.0V, the second column gives the energy consumed by the processors running at 3.3V and the third column gives the energy consumed by the processors running at 2.4V. The fourth column under each algorithm gives the total energy consumed by the system to execute a particular benchmark program. This column is given with a heading of “Total”.

Figure 21 illustrates the total energy consumed by the multiprocessor system to execute the benchmarks with varying number of nodes. From the figure, we can conclude that PbS algorithm consumes highest energy followed by E-PbS algorithm. SaS and SbS
algorithms consume lesser energy with SbS being higher than SaS algorithm.

![Energy Comparison Chart]

**Fig 21 Energy Comparison of Number of Tasks vs. Energy Consumed**

The energy consumed by the processors at different voltages for SaS and SbS algorithms is shown in charts in figure 22 and 23. Both the algorithms follow a similar trend for the most part of the graph. The energy consumed by the processors running at 2.4V is higher than the processors running at 5 and 3.3V. Among these two algorithms, the SaS algorithm consumes lesser energy and also the least among all the proposed algorithms. From the graphs, we can conclude that these two algorithms constantly try to execute as many nodes as possible at a lesser voltage.
Unlike SaS and SbS algorithms, PbS and E-PbS algorithms consume high energy. This can be seen in energy vs. number of tasks graph depicted in figure 24 and figure 25.
The reason for consumption of higher energy by the system is that more number of processors are running at 5.0V. Since, the energy consumed is quadratically proportional to the voltage of execution, the energy consumed increases.

Fig 24 Energy consumed per each voltage in PbS Algorithm

Fig 25 Energy consumed per each voltage in E-PbS Algorithm
Also, the main objective of PbS and E-PbS is to reduce the number of resources being utilized and there is a trade off between energy and resources in the process.

The utilization of each processor implementing the algorithms is tabulated in table 9.

Table 9 Average utilization of the processors

<table>
<thead>
<tr>
<th>Bench mark</th>
<th>Nodes</th>
<th>SaS</th>
<th>5.0V Util</th>
<th>3.3V Util</th>
<th>2.4V Util</th>
<th>SbS</th>
<th>5.0V Util</th>
<th>3.3V Util</th>
<th>2.4V Util</th>
<th>PbS</th>
<th>5.0V Util</th>
<th>3.3V Util</th>
<th>2.4V Util</th>
<th>E-PbS</th>
<th>5.0V Util</th>
<th>3.3V Util</th>
<th>2.4V Util</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.stg</td>
<td>7</td>
<td>1</td>
<td>0.993976</td>
<td>1</td>
<td>0.616162</td>
<td>3</td>
<td>0.437243</td>
<td>2</td>
<td>0.666667</td>
<td>2</td>
<td>0.804348</td>
<td>2</td>
<td>0.666667</td>
<td>3</td>
<td>0.777778</td>
<td>3</td>
<td>0.777778</td>
</tr>
<tr>
<td>10.stg</td>
<td>10</td>
<td>1</td>
<td>0.981745</td>
<td>1</td>
<td>0.2886</td>
<td>4</td>
<td>0.511474</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.8357</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.666667</td>
</tr>
<tr>
<td>50.stg</td>
<td>50</td>
<td>2</td>
<td>0.763864</td>
<td>1</td>
<td>0.696456</td>
<td>13</td>
<td>0.329297</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.8357</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.666667</td>
</tr>
<tr>
<td>sparse.stg</td>
<td>96</td>
<td>1</td>
<td>0.634146</td>
<td>1</td>
<td>0.314815</td>
<td>62</td>
<td>0.328045</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.8357</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.666667</td>
</tr>
<tr>
<td>100.stg</td>
<td>100</td>
<td>3</td>
<td>0.816992</td>
<td>5</td>
<td>0.516588</td>
<td>46</td>
<td>0.433302</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.8357</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.666667</td>
</tr>
<tr>
<td>300.stg</td>
<td>300</td>
<td>1</td>
<td>0.953523</td>
<td>0</td>
<td>0</td>
<td>120</td>
<td>0.320016</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.8357</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.666667</td>
</tr>
<tr>
<td>500.stg</td>
<td>500</td>
<td>6</td>
<td>0.397026</td>
<td>4</td>
<td>0.298212</td>
<td>205</td>
<td>0.309451</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.8357</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.666667</td>
<td>2</td>
<td>0.666667</td>
</tr>
</tbody>
</table>
One of the main parameters that signify the efficiency of the scheduling algorithms is the utilization of the resources. The energy table and graphs indicate that PbS and E-PbS algorithms consume very high energy compared to SaS and SbS algorithms. The average utilization of the processors running at various voltage levels executing the standard benchmarks used in this thesis are tabulated in table9. The first column in the table gives the names of the benchmarks used to simulate the program. The second column gives the number of nodes in each benchmark program. The four algorithms are shown in four different colors starting with SaS, SbS and followed by PbS and E-PbS. The table in each algorithm is divided into six columns. The first column shows the number of processors working at 5.0V. The second column gives the average utilization of all those processors. The third column gives the number of processors running at 3.3V and their average utilization is given in column four. Similarly, the fourth and fifth columns give the number of processors and their average utilization at 2.4V. From this table, we can conclude that processors in E-PbS algorithm have higher utilization factor. This shows that the algorithm is more efficiently employing all the processors to find a feasible schedule. This is again proved in figure 26 in which it is clearly seen that E-PbS utilizes the processor efficiently.
Fig 26 Comparison of Processor utilization vs. Number of tasks

Fig 27 Comparison of Processor utilization at 5.0V for each algorithm
Fig 28 Comparison of Processor utilization at 3.3V for each algorithm

Figures 27, 28 and 29 compare the utilization of the processors by Sas, SbS, PbS and E-PbS algorithms at 5.0V, 3.3V and 2.4V. In all the cases, it can be easily concluded that E-PbS and PbS have a higher utilization of the processors than SaS and SbS algorithms.

Fig 29 Comparison of Processor utilization at 2.4V for each algorithm
By comparing the number of processors required and energy consumed by the processors, we can conclude that SaS and SbS consume lesser energy while PbS and E-PbS use reduced number of resources to schedule the task sets.

Fig 30 Average Utilization of the processors expressed in percentage

Fig 31 Comparison chart for Energy consumed per processor
The graphs studied till now compare the scheduling algorithms on parameters that do not depict the efficiency of the algorithm. The processor utilization graphs show that E-PbS algorithm is highly successful in utilizing the processors. Figure 31 and figure 33 compare the four algorithms on the basis of a parameter which shows the efficiency of the algorithm. A comparison of energy per processor tells us that SaS consumes least energy per processor while executing the task set. The final graph compares the product of the energy and utilization of all the algorithms. An algorithm is efficient if the energy consumed by it is less and its utilization is high. So, the lesser the product of energy and 1/utilization, the better the algorithm. From figure 33, we can observe that PbS and E-PbS fare well in this front.
Comparison of Energy - Utilization

Fig 33 Graph showing the product of Energy and Utilization

Comparison of Energy * Deadline

Fig 34 Bar graph depicting the efficiency of each algorithm
5.5 Directions for future work

With scheduling in RTES being one of the challenging problems, the work done as part of this thesis leaves us with many unexplored areas. The algorithms implemented work for systems with as many as 1000 nodes. These algorithms can be extended to systems with even higher number of tasks.

We assume that the processor implementing the SaS, SbS, PbS and E-PbS algorithms can operate at one particular voltage and frequency. It will be interesting to see how the algorithms perform on processors which can operate at different voltage levels and frequencies. Processors with voltage shift ability are available in the market and can be used to simulate the algorithms.

The four algorithms developed in this algorithm minimize the energy and resources by scaling the voltage of operation. Including the frequency scaling in the algorithms would be an interesting area of research. We expect the energy consumed to reduce further by scaling the frequencies of operation.

Further, these scheduling algorithms can be implemented in a multiprocessor architecture to observe their real time performance. Another area of interest is to schedule task sets that have communication cost and delay between the tasks. Also, the tasks are assumed to be aperiodic and dependent. Further research is possible in the area of implementing the proposed algorithms in this thesis for periodic task sets.
REFERENCES


[9]. Loïc Pontani, Denis Dupont, “Scheduling and Assignment for Real-time Embedded


2003, pp. 211 – 218, 2-4 July 2003


Real Time Embedded Systems”, Dept. of Computer Science, Texas A&M University, College Station, TX.


VITA

Graduate College
University of Nevada, Las Vegas

Naveen Babu Anne

Local Address:
969 East Flamingo Road
Apartment 122
Las Vegas, Nevada 89119

Home Address:
Quarter 4 Agricultural Research Station
Ragolu, Srikakulam
Andhra Pradesh, India 532484

Degrees:
Bachelor of Engineering, Electrical Engineering, 2002
Malaviya National Institute of Technology, Jaipur
University of Rajasthan

Publications:
1. Three and Four-dimensional Parity-check Codes for Correction and Detection of Multiple Errors, ITCC 2004, Las Vegas
2. Branch Prediction by Checking Loop Terminal Conditions, ISNG 2005, Las Vegas

Thesis Title: Scheduling of real time embedded systems for resource and energy minimization by voltage scaling

Thesis Examination Committee:
Chair person, Dr. Venkatesan Muthukumar, Ph.D.
Committee Member, Dr. Henry Selvaraj, Ph.D.
Committee Member, Dr. Emma Regentova, Ph.D.
Graduate Faculty Representative, Dr. Ajoy K. Datta, Ph.D

83

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.