ENERGY-OPTIMAL SCHEDULES OF REAL-TIME JOBS WITH HARD DEADLINES

A Thesis
by
JOHN V. GEORGE

Submitted to the Office of Graduate Studies of Texas A&M University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee, Riccardo Bettati
Committee Members, Rabi N Mahapatra
Norman Guinasso
Head of Department, Valerie E Taylor

August 2005

Major Subject: Computer Science
ABSTRACT


John V. George, Bachelor of Technology, Mahatma Gandhi University;
Chair of Advisory Committee: Dr. Riccardo Bettati

In this thesis, we develop algorithms that make optimal use of frequency scaling
to schedule jobs with real–time requirements.

Dynamic voltage scaling is a technique used to reduce energy consumption in
wide variety of systems. Reducing supply voltage results in a lower processor clock
speed since the supply voltage has a proportional dependency on the clock speed of
the processing system.

In hard real–time systems, unduly reducing the speed of processor could result
in jobs missing their deadlines. The voltage scaling in such systems should therefore
take into consideration the deadline of jobs. This thesis will address two questions:
First, given a set of discrete frequency levels, we determine an energy-optimal sched-
ule of a given set of real-time jobs. We model the problem as a network flow graph
and use linear programming to solve the problem. The schedule can be used on
processors with discrete frequencies (like Transmeta Efficeon Processor and AMD
Turion 64 Processor).

Second, given a set of real–time jobs, we determine a set of optimal frequency
levels which minimizes the energy consumption while meeting all the timing con-
straints. This can be used to model variable-capacity facilities in operations re-
search, where the capacity of the facility can be controlled at a cost.
DEDICATION

To my Family and Dr. Norman L. Guinasso, Jr.
ACKNOWLEDGMENTS

God, Thy deeds are so perfect...

I would like to express my sincere gratitude to each and every person who has made this thesis possible.

My adviser, Dr. Riccardo Bettati, has been very patient, friendly and helpful all through the thesis. I am amazed by his brilliant ideas and approach towards problems. He has been an influence not only on my technical approach but also on the way I look at life. It is difficult to overstate my gratitude to him. Dr. Bettati, thank you so much.

I would like to thank Dr. Norman for his support throughout my stay at Texas A&M University. It would not have been possible to fulfill this dream without the financial and moral support of Dr. Norman. Thank you, Dr. Norman.

I express my thankfulness to Dr. Rabi for his suggestions and ideas all throughout the thesis.

I express my sincere thanks to Mr. Homarjun Agrahari and Mr. Aravind Aluri for their technical tips throughout the thesis. Mr. Paul Jensen, The University of Texas at Austin, helped me in using his ORMM software to simulate algorithms presented in this thesis. Thank you, Mr. Paul.

Without the support of my family and friends, I would never have reached this point in my life. I lack words to express my gratitude to my family and friends for their moral support.
TABLE OF CONTENTS

CHAPTER | Page
---------|-------
I | INTRODUCTION | 1
II | RELATED WORK | 4
III | DEFINITIONS AND SYSTEM MODEL | 7
   | A. Workload Model | 7
   | B. Processor Model | 8
   | C. Power Model | 9
   | D. Energy Aware Scheduling | 10
IV | PRELIMINARY OBSERVATIONS | 12
V | ENERGY OPTIMAL SCHEDULES ON DISCRETE FREQUENCY PROCESSORS | 14
   | A. Min-Cost Max-Flow Network Flow Problem | 14
      | 1. Network Flow Problem | 14
      | 2. Min-Cost Max-Flow Network Flow Problem | 15
      | 4. Algorithm to Create Min-Cost Max-Flow Graph | 19
      | 5. Proof of Correctness | 21
      | 6. Performance | 22
         | a. Graph Complexity | 22
         | b. Algorithm Complexity | 23
   | B. Linear Programming Formulation of the Discrete Frequency Processor | 23
VI | ENERGY OPTIMAL SCHEDULES ON CONTINUOUS FREQUENCY PROCESSORS | 26
   | A. Initial Approach | 27
      | 1. Continuous Frequency Problem Evaluation | 28
   | B. Jobs with Identical Release Time | 29
      | 1. Algorithm and Description | 29
      | 2. Proof of Correctness | 32
      | 3. Performance | 37
<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C. Jobs with Arbitrary Release Time</td>
<td>38</td>
</tr>
<tr>
<td>1. Energy Optimal Scheduling Through Extensive Frequency Enumeration</td>
<td>39</td>
</tr>
<tr>
<td>2. Implicit Frequency Enumeration</td>
<td>41</td>
</tr>
<tr>
<td>3. Algorithm and Description</td>
<td>43</td>
</tr>
<tr>
<td>4. Proof of Correctness</td>
<td>47</td>
</tr>
<tr>
<td>5. Performance</td>
<td>49</td>
</tr>
</tbody>
</table>

| VII | RESULTS | 50 |
| VIII | CONCLUSIONS AND FUTURE DIRECTIONS | 58 |

| REFERENCES | 60 |

| VITA | 63 |
## LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Processor models used in our experiments</td>
</tr>
<tr>
<td>2</td>
<td>Energy consumption of discrete, GNF and base</td>
</tr>
<tr>
<td>3</td>
<td>Percentage variation of energy consumption in discrete, GNF and base</td>
</tr>
<tr>
<td>4</td>
<td>Arcs and nodes of discrete versus GNF</td>
</tr>
<tr>
<td>5</td>
<td>Energy consumption of discrete and continuous on 5-(T) set</td>
</tr>
<tr>
<td>6</td>
<td>Energy consumption of discrete and continuous on 20-(T) set</td>
</tr>
<tr>
<td>7</td>
<td>Energy consumption of discrete and continuous on 10-(T) set</td>
</tr>
<tr>
<td>8</td>
<td>Variation of energy consumption of discrete, continuous and base</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Network flow graph representation</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>Schedule when $n = 2$</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>Schedule when $n = 3$ and $f_1 &gt; f_2$</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>Schedule when $n = 3$ and $f_2 &gt; f_1$</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>Schedule when $n = k + 1$ and $f_1 \leq f_2 \leq \ldots \leq f_k$</td>
<td>37</td>
</tr>
<tr>
<td>6</td>
<td>Schedule when $n = k + 1$ and $f_1 \geq f_2 \geq \ldots \geq f_k$</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>Flow graph representation of frequencies</td>
<td>39</td>
</tr>
<tr>
<td>8</td>
<td>Stage wise representation of Algorithm 5</td>
<td>45</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

In this thesis, we present algorithms that make optimal use of frequency scaling to schedule jobs with real-time requirements.

A job is said to have real-time requirements (or simply said to be real-time) if it has to finish within a certain time limit. Such jobs are often common in embedded systems where they arise from the interaction of the system with the environment. Reducing energy consumption in systems with real-time jobs has been a topic of research since over a decade. Dynamic Voltage Scaling (DVS) is a technique used to reduce energy consumption in real-time embedded systems. Energy consumption has a quadratic dependency on supply voltage and DVS aims at reducing energy consumption by reducing the processor’s supply voltage in tandem with its frequency. For the case of real-time jobs, DVS makes use of the fact that there is no benefit in finishing a job earlier than its deadline. Transmeta Efficeon Processor [1] and AMD Turion 64 [2] are examples of processors that can operate at several discrete operating frequencies based on load requirements.

Variable-voltage processors and algorithms, typically based on DVS, have been developed for periodic, aperiodic, and sporadic jobs. Reducing supply voltage results in a lower processor clock speed since the supply voltage has a proportional dependency on the clock speed of the processing system.

In systems with real-time jobs, unduly reducing the speed of the processor could result in one or more jobs missing their deadline. The voltage scaling in such

This thesis follows the style of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.
systems should therefore take into consideration timing constraints of jobs.

In this paper, we focus on generating energy efficient schedule for jobs with hard real-time requirements. The algorithms described in this thesis generate schedules offline, that is, before the execution of the jobs. This thesis addresses two questions:

First, given a set of discrete frequency levels and a set of jobs, we determine an energy-optimal schedule: Among all the feasible schedules possible an energy optimal schedule forms the lower bound on the energy consumption of the given job set.

We develop an algorithm to generate energy optimal schedules, and prove its optimality. That is, we show that the algorithm never fails to find an energy optimal schedule if such a schedule exists. The function to generate an optimal, feasible schedule is modelled as a minimum cost, maximum flow network flow graph problem. Since this form of network flow problems can be solved in quadratic time, our technique scales well to real-life job set.

Second, given a set of real-time jobs, we determine a set of frequency levels that minimizes the energy consumption while meeting the deadline constraints.

We evaluate the algorithms using randomly generated task sets and CNC, a real-life benchmark. We also show that the schedules generated by our algorithms consume less energy while taking polynomial time to generate such a schedule.

Note: Throughout this thesis, we assume that the energy consumption is super-linear with respect to the frequency levels of the processor, that is, \( E > O(V) \). This is the case in most architectures where \( E \propto V^2 \), where \( E \) is the energy consumed, and \( f \), the frequency level of the processor.

This thesis is organized as follows: In Chapter II, we give an overview on recent and current research work on DVS that other authors have proposed. The
definitions and system model used in this paper are described in Chapter III. A set of preliminary observations are stated in Chapter IV. Algorithms to generate optimal schedules are described in Chapters V and VI.
CHAPTER II

RELATED WORK

Different scheduling policies and various task models have been considered so far for Dynamic Voltage Scaling (DVS) of Real-Time Systems. DVS has been applied to both soft real-time jobs as well as hard real-time jobs.

The schemes proposed in [3] [4] [5] are based on a two-phase scheduling algorithm: Before run time, voltage settings are picked to reduce energy consumption based on workload assumptions. During run time, an online schedule adjusts voltage based on the resources that remain unused (like when jobs complete early). The authors in [4] proposes dynamic workload adaptation to take advantage of unused computation time that results from the variation of execution time of real-time tasks. A reward based approach to power aware scheduling is used to find an optimal static solution based on worst case workload. Dynamic reclamation is performed by adapting to the actual workload. The paper further proposes an aggressive speed reduction technique that can be employed to gain more energy savings by assuming a work load less than the worst case workload.

A novel slack estimation heuristic method to create an energy efficient DVS is used in [6]. The paper proposes an algorithm that estimates slack times more efficiently with a small additional overhead. The authors in [7] proposes an efficient method to handle DVS, by creating pseudo operating frequency levels between discrete frequency levels supported by the system. The energy efficient trade-off on a DVS enabled real-time system when the workload includes aperiodic jobs as well as periodic tasks is studied in [5]. The paper explores the performance of a system that consists of both periodic as well as aperiodic jobs. A composite metric Energy
× Average Response Time is used as a performance measure. A static algorithm and slack re-use scheme is also proposed in the paper.

Four voltage-scaling algorithms suitable for different system characteristics are proposed in [8]. An optimal frequency grid that minimizes the effect of discrete operating frequencies is derived in the paper. A formula for improving an existing DVS algorithm without any negative effect on the performance of the algorithm is proposed in [9].

An inter-task DVS for sporadic task model in conjunction with preemptive EDF scheduling is presented in [10]. Three voltage scaling schemes are proposed to schedule jobs with non-preemptive sections in [11]. Negative impacts on task scheduling and system wide energy can be reduced by controlling task preemption according to [12] [13]. An $O(N^2)$ algorithm, where $N$ is the number of jobs, to obtain minimum constant speed for each job and an $O(N^3)$ to obtain the minimum constant speed for the whole job set is presented in [14].

The authors of [15] formulate DVS for a set of periodic jobs as a nonlinear optimization problem and propose an optimal off-line algorithm to solve this problem. They further propose an online dynamic algorithm to reclaim slack cycles generated by early completion of jobs. In [16], a model of dynamically variable voltage processor is presented. Static voltage scheduling problem is formulated as an integer linear programming. A set of basic theorems for power-delay optimization is also presented in the paper.

The first algorithm presented in this paper is most closely related to [17]. It proposes two offline DVS schemes to minimize the energy consumed by a processor. The first solution models DVS problem as a generalized network flow (GNF) graph and uses the generalized network simplex algorithm to solve it. A GNF model cannot scale to large task sets using processor model with large number of operating
voltages. Thus, a drawback of the solution proposed in [17] is that it cannot scale to large number of frequencies. The second scheme discussed in [17] eliminates this drawback by developing a near-optimal minimum-cost network flow model.

The second drawback of [17] is that it assumes voltage switching only at task boundaries. This is called inter-task scheduling. The authors in [18] and [19] discuss how intra-task scheduling (that is, when voltage switching can occur also during the execution of a task) always results in greater energy savings than inter-task scheduling. The method discussed in [18] though efficient, have significant impact on the portability of the application, as it requires modifications to the operating system in most cases. An improvement on the method (based on worst-case execution time) in [18] is proposed in [19] (based on average-case execution time).

In this thesis, we concentrate on developing algorithms that are simple to implement and have the added advantages of intra-task scheduling. Unlike, many of the previously discussed algorithms we do not use check points or other complex schemes that would make the algorithm complex or non-portable. We also make sure that the algorithms works independent of the number, or lack there, of frequencies or jobs.
CHAPTER III

DEFINITIONS AND SYSTEM MODEL

In the following section, we describe a reference model for real-time systems used in this thesis. According to this model, each system is characterized by three elements: (1) a workload model [20] that describes the type of applications supported by the system, (2) a processor model that describes the processing resources available to the applications, and (3) a power model that defines the power consumed when an application executes on a processing unit.

A. Workload Model

The workload model used in this thesis captures only the timing behavior of the applications in the system. Other characteristics such as functional behavior are ignored. In this paper we consider a workload that consists of a set of \( n \) jobs \( J = \{J_1, J_2, \ldots, J_n\} \). Each job \( J_i \) is defined by its release time \( r_i \), its deadline \( d_i \), and the number \( x_i \) of CPU cycles required to execute the jobs. **Release Time** of a Job \( J_i \) is the time at which it is released. **Deadline** of a Job \( J_i \) is the time at or before which \( J_i \) should complete execution. Deadline of a job is always greater than its release time. We assume that a job can start its execution at any time between its release time and deadline. The number of cycles required by a Job \( J_i \) to complete its execution is the execution cycles. We represent each job \( J_i \) by a tuple \((r_i, d_i, x_i)\). We assume that a job can start its execution at or after its release time. We also assume that a job can be preempted at any time, and each job is independent of each other.

We call a task as **periodic task** if it executes repeatedly at regular or semi-
regular time intervals in a continuing basis. A task is called *aperiodic task* in this paper if it has no deadline and the inter arrival times between consecutive jobs vary widely. Tasks with inter arrival times that vary widely but have hard deadlines are called as *sporadic jobs*. An executable-interval of a Job $J_i$ is the set of intervals occurring between $r_i$ and $d_i$.

This workload model is very general in practice. For example, periodic workloads in embedded systems are often scheduled by indefinitely repeating a schedule laid out for the jobs during a so-called *hyperperiod*, which is the least common multiple of all the periods of the task set. We take a hyperperiod, $H$, of the jobs such that no job overlaps with any other job outside its hyperperiod. $H$ is assumed to have $n$ jobs.

B. Processor Model

Jobs need a processor in order to make progress towards completion. In this thesis, we assume two types of processors. We model each type of processor separately for the sake of convenience.

In the *Discrete Frequency Model*, we assume that the processor can execute at a frequency $f_k$ if and only if $f_k \in \mathcal{F} = \{f_0, f_1 \ldots f_m\}$, where $\mathcal{F}$, is the set of all the frequencies that can be used by the processor, sorted in increasing order. Thus, $f_0$ is the minimum frequency possible by $P$ and $f_m$ the maximum frequency possible by $P$. Throughout this thesis, we assume $f_0$ to be equal to 0. We call a processor that adheres to the discrete frequency model a *discrete frequency processor*.

In *Continuous Frequency Model*, we assume that the processor can execute at any frequency in the scale ranging from $f_0$ to $f_m$, where $f_0$ is the minimum frequency available to the processor, and $f_m$ is the maximum frequency available to
the processor. We call a processor that adheres to the continuous frequency model a continuous frequency processor.

C. Power Model

According to [21], CMOS circuits have both static and dynamic power dissipation. Static power arises from bias and leakage circuits. The reduction in the use of static gates as well as efficient design of modern gates have led to negligible power consumption in static gates.

Therefore the dominant part in most designs is the dynamic power consumption.

From [21], dynamic power consumption can be defined in simple equation as follows:

\[ P \approx V_{dd}^2 \times f_{clk} \times C_{eff}, \]  

(3.1)

where \( V_{dd} \) is the supply voltage, \( f_{clk} \) is the clock frequency, and \( C_{eff} \) is expressed as a product of physical capacitance, \( C_l \), and the activity weighting factor, \( \alpha \).

Since the energy consumption is equal to the power dissipation over time, we have

\[ Energy \approx V_{dd}^2 \times f_{clk} \times C_{eff} \times t \]  

(3.2)

where \( t \) is the time for which \( f_{clk} \) is used.

According to [18], the number of cycles, \( X_{cyc} \) executed in a processor, during time \( t \) at frequency \( f_{clk} \) is

\[ X_{cyc} = f_{clk} \times t. \]  

(3.3)

From Equation (3.3), we get

\[ Energy \approx V_{dd}^2 \times C_{eff} \times X_{cyc}. \]  

(3.4)
According to [10]

\[ t_d = k \frac{V_{dd}}{(V_{dd} - V_t)^2}, \quad (3.5) \]

where \( t_d \) is the circuit delay time, \( k \) a constant, and \( V_t \) the threshold voltage and

\[ f_{clk} = \frac{1}{L_d t_d}, \quad (3.6) \]

where \( L_d \) is the depth of the critical path[10].

Combining Equations (3.4), and (3.5) we get,

\[ \text{Energy} \propto V_{dd}^2 \times X_{cyc}. \quad (3.7) \]

Note: Equations (3.5) and (3.6) tell us that voltage is proportional to frequency, and frequency is inversely proportional to time delay. Thus, by reducing voltage, the frequency decreases, causing time delay to increase.

D. Energy Aware Scheduling

The goal of this thesis is to develop algorithms that, given a job set \( \mathcal{J} \), and a processor (discrete or continuous frequency), determine a schedule of execution of the jobs in \( \mathcal{J} \) and a set of frequency allocations in order to meet the job’s timing requirements (release time and deadlines) while minimizing the total energy consumption.

We say that a schedule is valid if it schedules all the jobs in \( \mathcal{J} \) in such a way that no two jobs are scheduled at the same time on a single processor.

A valid schedule is feasible where all the jobs in \( \mathcal{J} \) are scheduled in such a way that no job misses its deadline. Among all the feasible schedules there is one or more schedules where the total power consumption is minimal. We call such a schedule an energy optimal schedule. We call the sequence of frequencies at which the processor
executes the jobs in the schedule, the *frequency allocation*. An *optimal frequency allocation* gives rise to the minimum energy consumption. Therefore energy optimal schedules have optimal frequency allocations.

The rest of this thesis is about algorithms that generate energy optimal schedules.
CHAPTER IV

PRELIMINARY OBSERVATIONS

We note that the model observed in Section (III-C) is not limited to the case of $P \propto V^2$ but to any case where $P > O(V)$. A number of simple preliminary observations hold in this case. We list them as follows.

In the following observations, we substitute $f$ for $V$ by combining Equations (3.5) and (3.6).

**Lemma 1** [16] Given a single Job $J = (r, d, x)$, the frequency allocation $f$ is optimal when it is constant and $f = \frac{x}{d-r}$.

**Corollary 1** Given two jobs with identical release time, $J_1 = (r, d_1, x_1)$ and $J_2 = (r, d_2, x_2)$, $d_2 > d_1$. Let $f_1 = \frac{x_1}{d_1-r}$ and $f_2 = \frac{x_2}{d_2-d_1}$ be two frequencies used by $j_1$ and $j_2$ respectively. Whenever $f_2 > f_1$, the energy consumed will be minimum if $f_2$ equal to $f_1$.

**Lemma 2** Given two jobs with identical release time, $J_1 = (r, d_1, x_1)$ and $J_2 = (r, d_2, x_2)$, $d_2 > d_1$. Let $f_1 = \frac{x_1}{d_1-r}$ and $f_2 = \frac{x_2}{d_2-r}$ be two frequencies used by $j_1$ and $j_2$ respectively. Whenever $f_2 < f_1$, the energy consumed will be minimum if the jobs execute at frequencies $f_1$ and $f_2$ respectively.

**Proof:** Let us assume that jobs $J_1$ and $J_2$ run at frequencies other than $f_1$ and $f_2$ defined in the Lemma 2. This can be done in three ways: Without loss of generality we assume $r = 0$ in the following proof.

1. Decrease Job $J_1$’s frequency below $f_1$.

   Job $J_1$ cannot run at frequency lower than $f_1$ since it would miss its deadline otherwise.
2. Increasing $J_2$’s frequency above $f_2$.

This would increase the overall energy consumption.

3. Decrease $J_2$’s frequency below $f_2$.

Let $E$ be the energy consumed by the processor while executing at frequencies $f_1$ and $f_2$. From Equation 3.7 we have,

$$E = f_1^2 x_1 + f_2^2 x_2.$$  \hspace{1cm} (4.1)

Substituting for $x_1$ and $x_2$ from Lemma 2 we get,

$$E = f_1^3 d_1 + f_2^3 (d_2 - d_1).$$  \hspace{1cm} (4.2)

Assume that we decrease $J_2$’s frequency by moving $\Delta y$ cycles from interval $[d_1, d_2]$ to interval $[r_1, d_1]$. The energy consumed $E'$ in this case is

$$E' = (f_1 + \Delta y)^3 d_1 + (f_2 - \Delta y)^3 (d_2 - d_1).$$  \hspace{1cm} (4.3)

Since $\Delta y > 0$,

$$(\Delta y)^2 + 3 f_1^2 + 3 f_1 \Delta y > 0.$$  \hspace{1cm} (4.4)

Combining Equations (4.3, 4.2, and 4.4),

$$f_1^3 d_1 + f_2^3 (d_2 - d_1) < (f_1 + \Delta y)^3 d_1 + (f_2 - \Delta y)^3 (d_2 - d_1)$$  \hspace{1cm} (4.5)

which is equivalent to $E < E'$. This proves Lemma 2.
CHAPTER V

ENERGY OPTIMAL SCHEDULE ON DISCRETE FREQUENCY PROCESSORS

In this chapter, we define a problem to schedule a set of jobs with arbitrary release time, arbitrary deadline, and arbitrary execution time, on a processor with discrete set of frequencies, in an energy optimal manner. According to [22], an iterative flow network can be used to find a feasible cyclic schedule if such a schedule exists. According to [20], this network representation can be generalized to find schedules of jobs with arbitrary release time, arbitrary deadline, and arbitrary execution time as shown in the next section.

A. Min-Cost Max-Flow Network Flow Problem

Network flow has been used in a variety of settings to generate optimal schedules, typically in the context of generating feasible schedules or schedules that minimize other execution measures, such as makespan. In this section we describe how we formulate the problem of energy-optimal scheduling on a discrete-frequency processor as a Min-Cost Max-Flow network problem.

1. Network Flow Problem

Given a directed graph $G = (V, E)$, where $V$ is the set of nodes, and $E$ is the set of directed edges. An edge between any two nodes in $G$ is represented by $(u, v)$ where $u, v \in V$. Each edge $(u, v) \in E$ has a capacity $c(u, v)$ associated with it. We say that $c(u, v) = \emptyset, \forall (u, v) \notin E$. Given two special nodes Source $s$, and sink $t$, the
goal is to find a Flow $f(u, v)$ to each edge such that,

1. $\forall u, v \in V, \ f(u, v) > \emptyset$
2. $\forall u, v \in V, \ f(u, v) \leq c(u, v)$
3. $\forall u, v \in V, \ f(u, v) = -f(u, v)$
4. $\forall u \in V - \{s, t\}, \ \sum_{v \in V} f(u, v) = \emptyset.$

The problem is to find an allocation of flows to edges so as to maximize the total flow from Node $s$ to Node $t$. In this form, the network flow problem can be solved by the Ford-Fulkerson method. Simple scheduling problems, such as preemptive scheduling of jobs with arbitrary release times[20], feasible execution times, and deadlines on a single processor can be solved using simple network flow. In the energy optimal scheduling addressed in this paper we device a solution to a more general network flow problem.


In this variation of the Network Flow Problem each flow on an edge is associated with a cost function $\sigma(u, v, f(u, v))$. In the following we use a linear cost function, where

$$\sigma(u, v, f(u, v)) = C_{u,v} \times f(u, v),$$

and $C_{u,v}$ is some edge specific constant. For this simple case, the Min-Cost Max-Flow Network Flow problem can be solved using Linear Programming [23].

In the following, we describe how to formulate the problem of generating an energy-optimal schedule of a set of jobs as a Min-Cost Max-Flow network flow problem. We will be using the following notation to reason about frequency levels and time intervals:

We call $f'_j$ the difference between the frequency $f_j$ and $f_{j-1}$, i.e.

$$f'_j = f_j - f_{j-1}, \quad j \in 1, 2, \ldots m.$$ 

In addition, we partition the time into intervals $I_k$ as follows:

Sort the release times and deadlines of all the jobs in the system. All the neighboring entries then form a sequence of adjoining intervals, which we call $I_1, I_2, \ldots$. Our objective function is to minimize the energy consumption given by Equation (3.7).

The above equation can be modelled as a Min-Cost Max-Flow graph with the edge cost as a linear function of flow as shown in Figure (1). Any algorithm (like Network Simplex algorithm) that can solve a Min-Cost Max-Flow problem can be used to solve the graph. We define a flow graph, $G = (V, E)$, where $V$ is the set of vertices, and $E$ is the set of directed edges as follows.

- $\forall (u, v) \in E$, we represent flow along edge $(u, v)$ as $\sigma(u, v)$.
- $\forall (u, v) \in E$, we represent the cost of flow along edge $(u, v)$ as $\sigma(u, v)$.

The graph contains the following vertices and edges.

1. job vertex $J_i$, $\forall J_i \in \mathcal{J}$, representing job $J_i$.
2. interval vertex $I_j$, $\forall I_j \in \mathcal{I}$, representing interval $I_j$.
3. frequency vertex $f_k$, $\forall f_k \in \mathcal{F}$, representing frequency $f_k$.
FIG. 1. Network flow graph representation
4. There exists two special vertices \( s \) and \( t \), which represent the Source and Sink of the Flow Graph respectively.

5. \( \forall J_i \in \mathcal{J}, \) there exists a directed edge \((s, J_i), c(s, J_i) = x_i, \ \sigma(s, J_i) = 0 \).

6. \( \forall J_i \in \mathcal{J}, \forall I_j \in \mathcal{I}, \) there exists a directed edge \((J_i, I_j)\) if \( J_i \) can be scheduled in the interval \( I_j \), \( c(J_i, I_j) = \infty, \ \sigma(J_i, I_j) = 0 \).

7. \( \forall I_j \in \mathcal{I}, \forall f_k \in \mathcal{F}, \) there exists a directed edge \((I_j, f_k), c(I_j, f_k) = I_j \times f'_k, \ \sigma(I_j, f_k) = f(I_j, f_k) \times V_k^2 \).

8. \( \forall f_k \in \mathcal{F}, \) there exists a directed edge \((f_k, t), c(f_k, t) = \infty, \ \sigma(f_k, t) = 0 \).

Flow along each edge of the graph can be explained as follows.

\( f(s, J_i) \) The flow along this arc corresponds to the number of execution cycles scheduled by the flow algorithm. This edge ensures that the maximum number of execution cycles scheduled by a job \( J_i \in \mathcal{J} \) overall the intervals is equal to \( x_i \).

\( f(J_i, I_j) \) This edge represents the interval over which \( J_i \) can be scheduled. The flow along this edge corresponds to the number of cycles scheduled in each interval. It ensures that a job is scheduled only during those intervals that lie between its release time and deadline.

\( f(I_j, f_k) \) This cost of flow along this edge corresponds to the energy consumed to schedule the job at frequency \( f_k \). The flow along this arc also gives the frequency sketch for each of the intervals. The objective function which minimizes the cost of maximum flow is implemented by this arc.

\( f(f_k, t) \) This arc ensures that outflow from the source node is equal to inflow to the sink.

\[
\sum_{i=1}^{n} x_i = \sum_{j=1}^{N} \sum_{k=1}^{m} I_j f_k. \tag{5.1}
\]
Since $(J_i, I_j)$ gives the number of cycles executed by $J_i$ during the interval $I_j$, and $(I_j, f_k)$ gives the frequency sketch along each interval $I_k$, the schedule can be obtained by combining these two information.

4. Algorithm to Create Min-Cost Max-Flow Graph

We now describe the steps of the algorithm depicted in Algorithm 1.

Step 1 All the release times and deadlines are sorted in increasing order and intervals are formed by partitioning the time into $2^n - 1$ intervals, where $n$ is the number of jobs.

Step 2 A graph $G$ is defined with vertices formed by $J, I,$ and $F$. Two special nodes Source and Sink are defined as the start and end nodes of the graph.

Steps 3-4 The capacity of $(I_j, f_k)$ is equal to the number of cycles available during $I_j$ at $f_k$. Since all the cycles until frequency $f_{k-1}$ would be used before $f_k$, the actual frequency available at $f_k$ would be equal to the $f_k - f_{k-1}$. Therefore, we define $f_k' = f_k - f_{k-1}$. Note that the cost of using frequency $f_k$ is still equal to $f_k^2$.

Steps 5-15 The edges of the graph $G$ are defined.

Step 16 We run a standard Simplex algorithm [23] that would generate a maximum flow which minimizes cost along flow.

Step 17 As a result of the above step, we get the frequency allocation of the energy optimal schedule. The schedule is generated in the next step.

Step 18 From the energy optimal frequency allocation, compute the energy optimal schedule as follows:
Algorithm 1 Discrete-Freq-Scheduler
1: Sort \( r_i \)'s and \( d_i \)'s in increasing order. Let \( \mathcal{I} \) be the set of non-overlapping intervals formed by the partition of time from the earliest release time to the latest deadline into at most \( N \) intervals.

2: Construct Graph \( G = (V, E) \) where \( V = \{s, \mathcal{J}, \mathcal{I}, \mathcal{F}, t\} \), is the set of vertices.

3: \textbf{for all} \( f_k \in \mathcal{F} \) \textbf{do}
4: \hspace{1em} \( f'_0 = 0, \quad f'_k = f_k - f_{k-1} \)
5: \textbf{for all} \( J_i \in \mathcal{J} \) \textbf{do}
6: \hspace{2em} Define a directed edge \((s, J_i)\), \( c(s, J_i) = x_i \), \( \sigma(s, J_i) = 0 \)
7: \textbf{for all} \( J_i \in \mathcal{J} \) \textbf{do}
8: \hspace{3em} \textbf{for all} \( I_j \in \mathcal{I} \) \textbf{do}
9: \hspace{4em} \textbf{if} \( J_i \) can be scheduled in the interval \( I_j \) \textbf{then}
10: \hspace{5em} Define a directed edge \((J_i, I_j)\), \( c(J_i, I_j) = \infty \), \( \sigma(J_i, I_j) = 0 \)
11: \textbf{for all} \( I_j \in \mathcal{I} \) \textbf{do}
12: \hspace{3em} \textbf{for all} \( f_k \in \mathcal{F} \) \textbf{do}
13: \hspace{4em} Define a directed edge \((I_j, f_k)\), \( c(I_j, f_k) = I_j \times f_k \), \( \sigma(I_j, f_k) = f(I_j, f_k) \times V_k^2 \)
14: \textbf{for all} \( f_k \in \mathcal{F} \) \textbf{do}
15: \hspace{1em} Define a directed edge \((f_k, t)\), \( c(f_k, t) = \infty \), \( \sigma(f_k, t) = 0 \)
16: \textbf{Solve the Min-Cost Max-Flow Graph using any existing algorithm.}
17: \textbf{Construct the Frequency Allocation}
18: \textbf{From Frequency Allocation construct the Energy Optimal Schedule for} \( \mathcal{J} \).
We observe that if there is a flow from \( I_x \) to \( f_y \), there must be flows from \( I_x \) to all \( f_z \), where \( f_z < f_y \). This is caused by the cost function, which causes the flows from \( I_x \) to saturate first for lower frequency (and therefore lower cost) nodes. Given this, the frequency allocation of \( I_x \) can be obtained by dividing the flow from \( I_x \) to \( f_y \) by \( f'y \), i.e. \( \frac{f(I_x,f_y)}{f_y} \). The number of cycles of each Job \( J_r \), \( x_r \), scheduled in the interval \( I_x \), can be obtained by the flow from \( J_r \) to \( I_x \), \( f(J_r,I_x) \). Given the frequency allocation of an interval and the number of cycles of each job executing during that interval, we can obtain the Energy Optimal Schedule by \textit{shaving off} parts of the interval as we allocate the jobs in the interval.

Therefore, the total time complexity of the algorithm is equal to \( O(n^3) \)[23]

5. Proof of Correctness

**Theorem 1** Min-Cost Max-Flow Graph produces an energy optimal schedule, if such a schedule exists.

***Proof***: To prove that Min-Cost Max-Flow Graph produces an energy optimal schedule, it needs to be proven that the objective function that the graph minimizes represents the energy consumption of the processor. It also needs to be proven that the algorithm maximizes the flow through \( G \), which in turn proves that \( G \) produces a feasible schedule whenever one exists.

According to Equation (3.7), the square of frequency times the number of execution cycles used at that frequency represents energy consumption of the system. Since more than one frequency is being used by the processor, calculating the energy consumption at each frequency level and then adding up gives:

\[
E = E_1 + E_2 + \ldots E_m
\]
where $E_k$ represents the energy consumed by the processor at a frequency $f_k$.

$$E_k = V_k^2(x_1 + x_2 + x_3 + \ldots x_i) \quad (5.3)$$

thus, proving that the minimization function represents Equation (3.7) which in turn represents the energy consumed by a processor.

The graph created here is identical to Network flow graph explained in Section (A). The only additional constituent is the set of nodes $F$, which represents the frequency used during the schedule. But, $\forall I_j \in I, \ I_j$ is linked to $f_k, \forall f_k \in F$ which in turn is linked to the $t$. Thus, it is equivalent to $I_j$ being linked directly to $t$. Thus, the graph produces an feasible schedule with minimal energy consumption, if such a schedule exists.

6. Performance

In this section we analyze the complexity of the graph as well as the algorithm to create the graph.

a. Graph Complexity

Since there are $n$ jobs and $m$ frequencies, the number of nodes in the graph is bounded from above as follows:
The graph will have $n$ job nodes corresponding to each job, $2n - 1$ interval nodes corresponding to each interval, and $m$ nodes corresponding to each frequency. Since there are two special nodes corresponding to the source and the sink, the total number of nodes are bounded from above by $3n + 1 + m$.

The upper bound on the number of arcs can be estimated as follows:
The graph will have $n$ arcs from $s$ to $J_i, i = \{1, 2, \ldots, n\}$. In the worst case, the number of arcs from the job node to the interval node is $n(2n - 1)$. The number
of arcs from the interval node to the frequency node is \( m(2n - 1) \). The number of arcs from the interval nodes to sink is equal to \( m \). Thus the total number of arcs is bounded by \( 2n^2 + 2nm \).

b. Algorithm Complexity

The performance of the algorithm can be estimated as follows.

**Step 1** In the worst case a good sorting algorithm takes \( O(n \log(n)) \).

**Steps 2-7** Since there are \( 2n - 1 \) intervals Step (3) will take \( O(n) \), and Step (5) will take \( O(n) \) as well.

**Steps 8-11** Since there are \( n \) jobs and \( 2n - 1 \) intervals, the worst case time complexity to define arc \((J_i, I_j)\) will be \( O(n^2) \).

**Steps 12-14** Since there are \( m \) frequencies and \( 2n - 1 \) intervals, Steps (11) through (13) will take \( O(nm) \).

**Steps 15-16** For \( m \) frequencies, Step (14) will take \( O(m) \).

**Step 17** According to [23], the worst case time complexity of a Min-Cost Max-Flow Algorithm is \( O(n^3) \).

Therefore the worst case time complexity of the algorithm is \( O(n^3 + mn) \).

B. Linear Programming Formulation of the Discrete Frequency Processor

Based on the network flow formulation described in Section V-A, we give an equivalent Linear Programming formulation, which can be used as alternative to the Network Flow formulation. This leads to more compact representation of the problem to feed into the Linear Programming Solver. We define the following
parameters to formulate the problem in the Discrete Frequency Case:

**Parameters**

\[ x_i \]  number of execution cycles required by Job \( J_i \)

\[ s_k \]  start time of interval \( I_k \)

\[ e_k \]  end time of interval \( I_k \)

**Decision Variables**

\[ x_{jk} \]  number of cycles used during interval \( I_k \) at frequency range \( f'_j \)

\[ y_{ik} \]  number of cycles used by job \( J_i \) during interval \( I_k \)

**Objective**

\[
\min E = \sum_{j=1}^{N} \sum_{k=1}^{m} V_j^2 x_{jk} \tag{5.4}
\]

The objective function minimizes the total energy consumption of the processor based on the power model defined in Section (III-C).

**Constraints**

\[
\forall J_i \in J, \sum_{k=1}^{N} y_{ik} = x_i \tag{5.5}
\]

**Constraint 5.5** ensures that the sum of all the cycles of \( J_i \), executed over all intervals \( I_k \in I \), is equal to the number of execution cycles, \( x_i \), required, by \( J_i \).

\[
\forall J_i \in J, \forall I_k \in I, \quad r_i y_{ik} \leq s_k y_{ik} \tag{5.6}
\]

\[
\forall I_k \in I, \forall J_i \in J, \quad d_i y_{ik} \geq e_k y_{ik} \tag{5.7}
\]

**Constraints 5.6 and 5.7** ensure that \( J_i \) executes only during those intervals that lie between \( r_i \) and \( d_i \).

\[
\forall I_k \in I, \sum_{i=1}^{n} y_{ik} \leq (e_k - s_k) f_m. \tag{5.8}
\]
**Constraint 5.8** ensures that no interval is assigned more execution cycles than it can hold over its entire length at the maximum frequency, $f_m$.

$$\forall I_k \in I, \quad \sum_{i=1}^{n} y_{ik} \leq \sum_{j=0}^{m} x_{jk}. \quad (5.9)$$

**Constraint 5.9** ensures that the sum of execution cycles used by all the jobs in a particular interval is equal to the sum of the cycles used during all the frequency ranges $f'_k$ at that interval.

The decision variables, $x_{jk}$ and $y_{ik}$, of Linear Programming formulation defined above are non-negative real numbers. As said before, the above model can be input into a Linear Programming Solver to obtain an Energy Optimal Schedule.
CHAPTER VI

ENERGY OPTIMAL SCHEDULES ON CONTINUOUS FREQUENCY PROCESSORS

In the previous chapter we developed algorithms for energy optimal scheduling of real-time jobs on a processor capable of running at any of one or more discrete frequencies.

In this section we expand these results to the case of continuous-frequency processors. A continuous-frequency processor is capable of running at any frequency between a lower bound $f_0$ and an upper bound $f_m$.\(^1\)

Continuous frequency processors can be used, for example, to model variable-capacity facilities in operations research, where the capacity of the facility can be controlled at a cost. We observed in Chapter IV that the frequency allocation on a continuous-frequency processor during energy-efficient operation is discrete. The additional degree of freedom added by continuous-frequency processors is the ability for the designer to freely choose the frequency levels at scheduling time. For static systems this means that the frequency allocation is performed at system design time.

In the context of processor scheduling, continuous-frequency processor scheduling therefore can for example be used to determine optimal discrete frequency settings for application specific processor realizations.

In this chapter, we define the problem to schedule, in an energy-optimal feasible

---

\(^1\)In the following we assume $f_0 = zero$ and $f_m$ to be the maximum frequency possible by the processor. In practice, any energy optimal scheduling algorithm will assign some maximum execution frequency. No feasible schedule exists if this frequency exceeds $f_m$. 
manner, a set of jobs with arbitrary release time, arbitrary deadline, and arbitrary execution time on a single processor that can assume any frequency between $f_0$ and $f_m$.

A. Initial Approach

A possible method to estimate an optimal frequency allocation for a given job set $J$ is to enumerate all the probable frequencies (based on Lemmas 1 and 2 and Corollary 1) that jobs in $J$ can execute. Once this set of frequencies are known, the continuous-frequency problem is reduced to a discrete-frequency problem described in Chapter V. Unfortunately, in practice the number of possible frequencies is too large for this approach to succeed as shown in the following.

An upper bound on the number of frequencies possible by all the jobs in $J$ can be calculated as follows:

- If we schedule jobs of subsets of size one of $J$, we need at most $C(n, 1)$ different frequencies.

- If we schedule jobs of subsets of size two, we need at most $C(n, 2)$ additional frequencies. These frequencies occur when the execution intervals of two jobs intersect. The frequency is calculated as the sum of execution cycles of both the jobs, divided by the difference of the latest deadline job with the earliest released job among both the jobs.

- Similarly, if we schedule jobs with subsets of size $n$ we need at most
\[
\sum_{i=1}^{n} C(n, i) \quad (6.1)
\]
additional frequencies.
If we schedule a subset of size one first, and then schedule the remaining $n-1$ jobs in subsets of size 1 of $J$, there are at most $C(n-1, 1)$ additional frequencies.

Similarly, there can be at most

$$\sum_{i=1}^{n-1} C(n - 1, i)$$

(6.2)

additional frequencies if we schedule the remaining $n - 1$ jobs of $J$ in subsets of size $n$.

Since there are $n$ jobs, the maximum number of possible frequencies is at most equal to

$$n \times \sum_{i=1}^{n-1} C(n - 1, i).$$

(6.3)

Adding up all the possible frequencies for $J$, in the worst case we get,

$$\sum_{i=1}^{n} C(n, i) + n \times \sum_{i=1}^{n-1} C(n - 1, i).$$

(6.4)

Thus, it can be seen from the above equation that frequency enumeration is exponential.

1. Continuous Frequency Problem Evaluation

In this section we evaluate the difficulty of generating an energy optimal schedule on a continuous-frequency processor. According to Equation (3.7), Energy is proportional to the square of Voltage times the Number of Cycles.

In discrete frequency model problem, the frequencies (and their respective voltages) at which a processor can be scheduled are given. Therefore, the problem to find an energy optimal schedule on a discrete frequency processor is linear. In a continuous-frequency model problem, voltages and the corresponding frequencies
are unknown and therefore the problem becomes non-linear and more difficult to solve.

In the next section, we solve an easier problem by assuming that all the jobs in \( J \) have identical release time. We then propose a solution in Section VI-C to schedule a set of jobs with arbitrary release times on a continuous-frequency processor.

B. Jobs with Identical Release Time

The scheduling problem addressed in this section can be formulated as follows: Given a set of independent, preemptive jobs \( J = \{J_1, J_2, \ldots, J_n\} \), each with identical release time \( r \), arbitrary deadline \( d_i \) and arbitrary execution time \( x_i \), schedule the jobs on a single processor with continuous frequencies in an energy optimal feasible manner.

In the following, we make use of the lemma’s presented in Chapter IV to propose an algorithm to schedule a set of jobs. We call the resulting algorithm Energy-EDF.

1. Algorithm and Description

Energy-EDF algorithm can be described as follows:

The jobs in \( J \) are sorted in ascending order based on their deadline. The execution order of the jobs are in Earliest Deadline First (EDF) manner. We optimize each job, starting with the job having the earliest deadline to the job having the latest deadline, using a greedy approach.

We call \( J_i \) a neighbor of \( J_i \) if they form neighboring entries in the sorted sequence of jobs. The first job \( J_1 \) in \( J \) is scheduled in the interval \([r_1, d_1]\). For each job \( J_i \in J, i \in \{2, 3, \ldots, n\} \) starting with the job having the earliest deadline to
the job having the latest deadline: if the frequency of allocating $J_i$ from $d_{\text{PREVIOUS}}$ (deadline of job $J_{\text{PREVIOUS}}$ - the neighbor of $J_i$ whose $d_{\text{PREVIOUS}} < d_i$) to $d_i$ is less than the frequency of $J_{\text{PREVIOUS}}$, $f_{\text{PREVIOUS}}$ then, $J_i$ is scheduled in the interval $[d_{\text{PREVIOUS}}, d_i]$; Else $J_i$ and $J_{\text{PREVIOUS}}$ are scheduled from the start time of $J_{\text{PREVIOUS}}$, $s_{\text{PREVIOUS}}$, to $d_i$ at a frequency equal to the sum of their execution cycles divided by the total allocated execution time ($d_i - s_{\text{PREVIOUS}}$).

Algorithm 2 ENERGY-EDF (identical release time)

Require: $J = \{J_1, J_2, \ldots, J_n\}$ where $J_i = (0, d_i, x_i)$

Ensure: Energy Optimal Schedule

1: Sort the jobs in $J$ on $d_i$ such that $d_1 < d_2 < \ldots < d_n$
2: Let $s_i$ be the start time of $J_i$ and $s_i = 0$.
3: Create empty Stack
4: for $i \leftarrow 1, n$ do Optimize($J_i$)

The algorithm to optimize a set of jobs $J$, is divided in to three functions. The main function, Algorithm 2, sorts the job set and calls Optimize Algorithm 3, to optimize the frequency allocation of each job. The third function, Algorithm 4 merges two jobs in to one.

The Algorithm 2 can be explained as follows.

Steps 1-3 $J$ is sorted in ascending order of deadlines of jobs. A empty Stack is created for later use and the start-time of all the jobs are initialized to their release time.

Step 4 For each job, starting with the earliest deadline job to the latest deadline job, Optimize is called to optimize each job.

Algorithm 3 can be described as follows:

A Stack is used to store the schedule created by the algorithm. $r_i$ is assigned the
**Algorithm 3** Optimize jobs

1: function Optimize(job \( J_i \))

2: if Stack is empty then

3: Schedule \( J_i \) in the interval \([0, d_i]\) at frequency \( f_i = \frac{x_i}{d_i - 0} \)

4: Push(\( J_i \)) to Stack

5: else

6: \( J_{P R E V I O U S} \leftarrow \text{Stack.Pop()} \)

7: \( s_i = d_{P R E V I O U S} \)

8: if \( \frac{x_i}{d_i - s_i} > f_{P R E V I O U S} \) then

9: \( J_{P R E V I O U S} = \text{Merge}(J_i, J_{P R E V I O U S}) \)

10: Optimize(\( J_{P R E V I O U S} \))

11: else

12: Push(\( J_{P R E V I O U S} \)) to Stack

13: Schedule \( J_i \) in the interval \([s_i, d_i]\) at frequency \( f_i = \frac{x_i}{d_i - s_i} \)

14: Push(\( J_i \)) to Stack

**Algorithm 4** Merge jobs

1: function Merge(\( J_i, J_{P R E V I O U S} \))

2: Create a temporary job \( J_{T E M P} \)

3: \( s_{T E M P} = s_{P R E V I O U S}, \ d_{T E M P} = d_i, \ x_{T E M P} = x_i + x_{P R E V I O U S} \)

4: Schedule \( J_{T E M P} \) in the interval \([s_{T E M P}, d_{T E M P}]\) at frequency \( f_{T E M P} = \frac{x_{T E M P}}{d_{T E M P} - s_{T E M P}} \)

5: Return \( J_{T e m p} \)
start time of $J_i$ in the algorithm. $J_i$ is scheduled from $d_{PREVIOUS}$ to $d_i$ if $f_i$ is less than $f_{PREVIOUS}$; else both jobs are scheduled from $r_{PREVIOUS}$ to $d_i$.

**Steps 2-4** If Stack is empty, then schedule $J_i$ in the interval $[r_i, d_i]$ with frequency $f_i = \frac{x_i}{d_i - r_i}$. Push $J_i$ to Stack and Return.

An empty stack means that the incoming job is either the first job or is obtained by merging all the jobs scheduled before the current time.

**Steps 6-7** Else If there exists at least one job in the stack then, pop it out to $J_{PREVIOUS}$. Assign the start time of $J_i$ to $d_{PREVIOUS}$.

**Steps 8-10** Let $f_i = \frac{x_i}{d_i - r_i}$. If $f_i > f_{PREVIOUS}$ then, $J_{PREVIOUS}$ and $J_i$ are scheduled from the start time of $J_{PREVIOUS}$ to $d_i$ at frequency $f_i$. Optimize function is called recursively with $J_{PREVIOUS}$ as its argument.

The process of scheduling two jobs at an identical frequency as in Steps [8-10] is done using a function called Merge. This function schedules the jobs as stated above and returns the scheduled job.

**Steps 12-14** If $f_i < f_{PREVIOUS}$ then, both $J_{PREVIOUS}$ and $J_i$ are pushed to stack separately.

Once Energy-EDF runs to completion, the schedule can be obtained by popping the stack. Such a schedule minimizes the energy consumed by the job set, as proven below.

2. Proof of Correctness

**Lemma 3** Any job scheducable by EDF is scheducable by Energy-EDF.

**Proof**: Let $J_i = (r, d_i, x_i)$ and $J_k = (r, d_k, x_k)$ be two neighboring jobs scheduled at
frequency $f_i$ and $f_k$ during intervals $I_i = [r, d_i]$ and $I_k = [d_i, d_k]$ using Energy-EDF Algorithm. The frequency can be allotted in two ways.

**Case 1:** $f_k \leq f_i$ In this case, Energy-EDF does not do any re-scheduling. It assigns $J_i$ in the interval $[r, d_i]$ and $J_k$ in the interval $[d_i, d_k]$. Since $d_i < d_k$, the algorithm schedules the jobs in an EDF manner.

**Case 2:** $f_k > f_i$ Here, Energy-EDF re-schedules $J_i$ and $J_k$ at a frequency $f = f_k = f_i = \frac{x_i + x_k}{d_k - r_i}$. Since, the frequency of execution of $J_i$ increases from $f_i$ to $f$, $J_i$ completes its execution before $d_i$. $J_k$ starts its execution at the time when $J_i$ completes its execution. Both the jobs are still scheduled in an EDF manner.

Without loss of generality, the above case of scheduling two jobs can be extended to a set of $n$ jobs as shown in the proof below. Thus any job schedulable by EDF is schedulable by Energy-EDF as well.

**Lemma 4** Given a set of $n$ jobs, $J = \{J_1, J_2, \ldots, J_n\}$ where $J_i = (r, d_i, x_i)$ sorted in the increasing order of their deadlines $d_i$, Energy-EDF schedules $J$ in an energy optimal manner.

**Proof:** Energy-EDF is a greedy algorithm that works on two-job optimization scheme. We call a scheme that can optimize the whole job set taking subsets of two jobs at a time as two-job optimization scheme.

**Number of jobs** $n = 1$

If there is only a single job $J_1$, Energy-EDF schedules $J_1$ at a frequency $f_1 = \frac{x_1}{d_1 - r_1}$.

According to Lemma 1, this is the optimal frequency to schedule a single job.

**Number of jobs** $n = 2$

Let $J_1$ and $J_2$ be the two jobs to be scheduled by Energy-EDF. There are two ways in which $J_1$ and $J_2$ can be scheduled by Energy-EDF.
$f_2 \leq f_1$: In this case Energy-EDF does not do any re-scheduling. $J_1$ and $J_2$ are scheduled at frequencies $f_1$ and $f_2$. According to Lemma 2, if $f_2 \leq f_1$, it is optimal to schedule $J_1$ and $J_2$ at frequencies $f_1$ and $f_2$ respectively. Therefore, Energy-EDF schedules the job in an energy optimal manner. This is shown in Figure 2(a).

$f_2 > f_1$: Energy-EDF schedules the jobs $J_i$ and $J_2$ at a frequency $f = \frac{x_1 + x_2}{d_2 - r_1}$. According to Corollary 1, the optimal frequency to schedule $J_1$ and $J_2$ if $f_2 > f_1$ is $\frac{x_1 + x_2}{d_2 - r_1}$. This is shown in Figure 2(b).

Thus when the number of jobs equal two, Energy-EDF generates an optimal schedule.

**Number of jobs $n = 3$**

When the number of jobs equal three, Energy-EDF algorithm will first generate a schedule for $J_1$ and $J_2$ before taking $J_3$ into consideration. $J_1$ and $J_2$ will be scheduled in an energy optimal manner among themselves by the time $J_3$ is considered by Energy-EDF.

There are only two ways in which Job $J_3$ will have its frequency with respect to $J_2$. 
FIG. 3. Schedule when $n = 3$ and $f_1 > f_2$

(a) $f_3 < f_2$  
(b) $f_3 \geq f_2$

FIG. 4. Schedule when $n = 3$ and $f_2 > f_1$

(a) $f_3 < f_2$  
(b) $f_3 \geq f_2$
$f_3 < f_2$: Jobs $J_1$ and $J_2$ can have their schedules in two ways as follows:

$f_2 < f_1$: It is already proven that $J_1$ and $J_2$ form an optimal schedule among each other. Since $f_3 < f_2$, $J_3$ and $J_2$ form an optimal schedule according to Lemma 2. This is shown in Figure 3(a).

$f_2 = f_1$: Since $J_1$ and $J_2$ merged together to form a single job of identical frequency, $J_3$ and $J_1$, $J_2$ can be scheduled using a two-job scheduling scheme. Since it is already proven that Energy-EDF generates an optimal schedule when the number of jobs equal two, the schedule is optimal in this case. This is shown in Figure 4(a).

$f_3 \geq f_2$: In this case $J_3$ and $J_2$ merge together to form a single job with equal frequency. Now, the three job scheduling problem is reduced to two job scheduling problem. In the previous section we already proved that Energy-EDF generates an optimal schedule when the number of jobs equal two. These scenarios are depicted in Figures 3(b) and 4(b).

Assume the algorithm generates an optimal schedule for $k$ jobs by reducing it into two-job scheduling problem.

**Number of jobs** $n = k + 1$

When there are $k+1$ jobs, $k$ jobs with earliest deadline are scheduled first. The schedule will look as shown in Figure (5) or (6).

1. $f_{k+1} < f_k$

   According to Lemma 2, $J_{k+1}$ and $J_k$ are scheduled in an energy optimal manner among themselves. Since \{\(J_1, J_2, \ldots, J_k\}\) already form an energy optimal schedule among themselves, the jobs in job set \{\(J_1, J_2, \ldots, J_{k+1}\)\} is also scheduled in an optimal manner.

2. $f_{k+1} > f_k$
Energy-EDF reschedules $J_k$ and $J_{k+1}$, such that both the jobs are merged to form a single job with equal frequency, thus reducing the number of jobs to be scheduled to $k$. It is already proven that Energy-EDF generates an energy optimal schedule for $k$ jobs.

Thus Energy-EDF generates energy optimal schedule based on a greedy scheme called two-job optimization.

3. Performance

The time complexity of the Algorithm 3 can be estimated as follows.

**Step 1** A sorting algorithm takes $O(n \log(n))$ in the worst case.

**Step 2** This step takes a constant time to execute

**Step 3** Since there are $n$ jobs, it would take $O(n)$ to complete.

The time complexity of the Algorithm 2 can be estimated as follows:

**Step 1** The function is executed $O(n)$ times from Algorithm 2. In the worst case,
it will be called $O(n)$ times by itself during each such call. Thus, this step takes $O(n^2)$ in the worst case.

**Step 2-8** All the other steps take a constant time to execute.

Algorithm 4 takes a constant time to execute and therefore the worst case execution time of *Energy-EDF* is bounded by $O(n^2)$.

In the worst case the algorithm will take $O(n)$ extra space to store the schedule in a stack.

In the next section we first propose an algorithm based on the assumption that frequency enumeration is possible. We then propose a method to cut down the number of frequencies to polynomial and use a similar method to solve the problem of scheduling a set of jobs with arbitrary release time.

**C. Jobs with Arbitrary Release Time**

The scheduling problem addressed in this section can be formulated as follows:

*Given, a set of independent, preemptive jobs with arbitrary release time, arbitrary*
deadline, and arbitrary execution time, deduce an algorithm to schedule the jobs on a single processor in an energy optimal feasible manner.

1. Energy Optimal Scheduling Through Extensive Frequency Enumeration

![Flow graph representation of frequencies](image)

**FIG. 7.** Flow graph representation of frequencies

In the following we assume that all the possible frequencies for every job in $J$, can be enumerated. Then, we propose a Dynamic-Programming (DP) based approach to obtain an optimal frequency allocation on a continuous-frequency processor model.
1. $F = \{ f_{i,j}, t \}$ is the set of all vertices in $G$, where the vertices $S_i = \{ f_{i,1}, f_{i,2} \ldots f_{i,k_i} \}$, are all the possible frequencies that $J_i$ could be executed. In the following, we call $S_i$ a stage in the networks.

2. An arc $(f_{i,j}, f_{i+1,l}) \in E$ if there exists a feasible schedule when $J_i$ and $j_{i+1}$ are executed at $f_{i,k}$ and $f_{i+i,l}$ respectively.

3. Cost of arc $(f_{i,j}, f_{i+1,l})$, $\sigma(f_{i,j}, f_{i+1,l})$ is $V_{i,k}^2 x_i$, where $V_{i,k}$ is the voltage corresponding to $f_{i,j}$.

4. $S_{n+1}$ is a special node called sink $t$.

By definition, any path connecting a node in $S_1$ to $t$, as shown in Figure (7), represents a feasible schedule in $J$. Among all the feasible schedules in $J$, the energy optimal schedule is a path which minimizes the sum of all the arc costs of the path from a node in $S_1$ to $t$.

For a given feasible frequency graph, the energy optimal schedule (i.e., the energy optimal path) can be determined as follows.

Denote the energy optimal path from $f_{i,q}$ to $t$ as $\Psi(f_{i,q})$. Then,

$$\Psi(f_{i,q}) = \begin{cases} 
0, & \text{if } i = n + 1; \\
\min\{\sigma(f_{i,q}, f_{i+1,r}) + \Psi(f_{i+1,r})\}, & \text{if } i = 1, 2, \ldots, n
\end{cases}$$

where node $f_{i,q} \in S_i$, and node $f_{i+1,r} \in S_{i+1}$.

The value $\sigma(f_{i,q}, f_{i+1,r})$ represents the cost of the arc from node $f_{i,q}$ to node $f_{i+1,r}$.

Though dynamic programming does give us an optimal solution for the above problem, the algorithm to generate all the possible frequencies of $J$ would still have an exponential time complexity.
Since frequency enumeration is exponential and the algorithm relies on the assumption that frequency enumeration is possible, we now propose three lemmas and a theorem to cut down the number of probable frequencies to polynomial. We then use an approach similar to the one presented in this section to generate a schedule on a continuous-frequency processor in polynomial time.

2. Implicit Frequency Enumeration

In this section, we present a set of lemmas and a theorem that we use to reduce the number of possible frequencies to polynomial in terms of number of jobs.

**Lemma 5** [16] Given a set of jobs $\mathcal{J}$, the energy consumed will be minimum only if each interval in the interval set $\mathcal{I}$ executes at a constant frequency.

**Lemma 6** [16] Given a set of jobs $\mathcal{J}$, the energy consumed will be minimum only if each job executes at a constant frequency.

**Lemma 7** Given a set of jobs $\mathcal{J}$, the energy consumed will be minimum if the frequency of execution of every job in $\mathcal{J}$ is equal to or lower than the frequency of execution of all the other jobs scheduled over its executable interval.

**Proof**: Assume that there is at least one other job executing with a lower frequency in the executable interval of Job $J_i$. We can always reschedule $J_i$ such that it executes at a lower frequency. By extending the proof of Lemma 2 it can be seen that the energy consumed while executing a job at lower frequency is lower than the energy consumed to execute a job at higher frequency, thus proving Lemma 7.

**Theorem 2** For a schedule to be energy-optimal, it is necessary and sufficient for it to satisfy Lemmas 5, 6 and 7.
Using the above theorem (proof given in Section VI-4), we propose a polynomial-time DP approach to calculate the optimal frequencies. We call the resulting algorithm \textit{Cont-Freq-Scheduler}. The jobs are initially sorted in the increasing order of their release times. An initial schedule is generated by scheduling the jobs from their release time either to the release time of the next job or to its own deadline, which ever comes earlier. Then, starting from job have the latest deadline to the job have the earliest deadline, we run an \textit{Optimize} algorithm to obtain an energy-optimal schedule.

A minimum allowable frequency is assigned to every interval. No interval can execute at a frequency lower than the minimum allowable frequency. This minimum frequency is initialized to zero and is updated as and when new jobs are scheduled in that particular interval. A frequency is considered \textit{relevant} if it is at least as high as the minimum allowable frequency and minimizes the energy consumption of all the interval sets.

At each state of a stage in DP, the algorithm tries to minimize the energy consumption of two sets of intervals, $F_m$ and $F_n$ by re-scheduling a job (having deadline equal to the end time of $F_n$) in $F_m$ to $F_n$ if the frequency of $F_m$ is higher than $F_n$. Thus, the exhaustive search is reduced to a scenario of finding a relevant frequency for two sets of intervals based on a single job.

Let $\mathcal{Z}$ represent a set of intervals and $r_{\mathcal{Z}}, d_{\mathcal{Z}}, t_{\mathcal{Z}}$ represent the \textit{start-time}, \textit{end-time} and the length of $\mathcal{Z}$ respectively. Let $x_{\mathcal{Z}}$ equals the maximum number of cycles executable in $\mathcal{Z}$ and $f_{\mathcal{Z}}$ the frequency of $\mathcal{Z}$. We use $t_{\mathcal{Z}i}$ and $x_{\mathcal{Z}i}$ to represent the time interval from $d_{\mathcal{Z}}$ to $d_i$ and the number of cycles in the interval $[d_i, d_{\mathcal{Z}}]$ respectively. We use $\mathcal{IS}_i$ to represent the set of intervals in which $J_i$ is scheduled initially, $\forall i, \mathcal{IS}_i = \mathcal{I}$. 
3. Algorithm and Description

**Algorithm 5** Cont-Freq-Scheduler (arbitrary release time)

**Require:** $\mathcal{J} = (r_i, d_i, x_i)$

**Ensure:** Energy optimal feasible schedule

1. Sort $r_i$ and $d_i$ in increasing order. Let $\mathcal{I}$ be the set of non-overlapping intervals formed by $r'_i$s and $d'_i$s.
2. for all $I_j \in \mathcal{I}$ do, $I_j.minima = \emptyset$
3. if $r_{i+1} < d_i$ then, schedule $J_i$ in $\mathcal{I}_S_i = \{r_i, r_{i+1}\}$
4. else schedule $J_i$ in $\mathcal{I}_S_i = \{r_i, d_i\}$
5. for $i \leftarrow n, 1$ do, Optimize($\mathcal{I}_S_i$)

A recursive method is used to solve the problem of generating an optimal schedule for a processor with continuous frequencies. The jobs are scheduled from its release time to either its own deadline or the release time of the next job, whichever comes earlier. Thus the job that is released last is scheduled from its release time to its deadline.

The algorithm uses a DP approach to create an energy optimal schedule. Algorithm starts from the job with the latest release time and goes on backward to schedule each job in an energy optimal manner. At each stage of the algorithm, the next job with the latest deadline is taken into consideration. At the end of any stage $S_i$, jobs considered at or after $S_i$ are scheduled in an energy optimal with respect to each other.

A stage in the algorithm consists of several states. In each of the state it re-schedules jobs in the interval sets being considered to minimize the energy consumption of the interval sets. The schedule at the end of each stage is shown in Figure 8. We make use of two functions to schedule the jobs optimally. The first
Algorithm 6 Optimize a given set of intervals

1: function Optimize(Interval Set $\mathcal{Z}$)

2: \hspace{1em} for all $J_i \in \mathcal{Z}$ do

3: \hspace{2em} Take the next job $J_i$, having earliest deadline after $d_Z$

4: \hspace{2em} Calculate a relevant frequency, $f_{\text{opt}}$, for interval sets $\mathcal{Z}$ and $\{d_i, d_Z\}$

5: \hspace{2em} if $f_Z$ cannot be minimized then, goto Step [6.3]

6: \hspace{2em} if additional execution cycles required to schedule $\{d_Z, d_i\}$ at $f_{\text{opt}} > (x_i \in \mathcal{Z})$ then

7: \hspace{3em} $x_i \in \mathcal{Z}$ are rescheduled to $\{d_i, d_Z\}$ and the $f_Z$ and frequency of $\{d_i, d_Z\}$ are recalculated.

8: \hspace{2em} Optimize($\{d_i, d_Z\}$)

9: \hspace{2em} else

10: \hspace{3em} Reschedule $J_i \in \mathcal{Z}$ to $\{d_i, d_Z\}$ such that both the interval sets execute at $f_{\text{opt}}$.

11: \hspace{2em} Update the minima of each interval in $\{r_Z, d_i\}$

12: \hspace{2em} Optimize($\{d_i, d_Z\}$)

13: \hspace{1em} for all $I_j \in \mathcal{Z}$ do

14: \hspace{2em} if $I_j.\text{minima} < f_Z$ then, $I_j.\text{minima} = f_Z$
FIG. 8. Stage wise representation of Algorithm 5
algorithm starts a stage in DP, while the second algorithm creates the states to obtain an optimal energy schedule for jobs considered up to the current state.

Algorithm 5 can be described as follows.

**Step 1** This step sorts the release times and deadlines of all the jobs. Non-overlapping intervals are formed by immediate release times and deadlines.

**Step 2** Initialize the minimum allowable frequency of each interval as zero.

**Steps 3-4** Jobs are scheduled from their release time to either their deadlines or to the release time of the next job, which ever occurs earlier.

**Step 5** Starting from the job that is released last to the job that is released first, intervals of each job are passed on to Algorithm 6.

Algorithm 6 takes a set of intervals as its input and minimizes the energy consumption of jobs scheduled at or after the current interval. The steps can be explained as follows.

**Step 1** An interval set $\mathcal{Z}$, is obtained in the function Optimize()

**Steps 2-3** A job, $J_i$ scheduled in $\mathcal{Z}$ and having the earliest deadline after the end time of $\mathcal{Z}$ is taken

**Steps 4-5** A relevant frequency $f_{opt}$ is the one which minimizes the energy consumption but does not go below the minimum allowable frequency of the two interval sets is calculated. The number of cycles required by $\{d_z, d_i\}$ to obtain $f_{opt}$ is also calculated. If the frequency of $J_i$ cannot be minimized then, we take the next such job.

**Steps 6-8** At each state only $J_i$ is re-scheduled from $\mathcal{Z}$ to $\{d_z, d_i\}$. If $x_i$ available in $\mathcal{Z}$ is less or equal to the required cycles at $\{d_z, d_i\}$, then we transfer as
much of $x_i$ available at $\mathcal{Z}$ to $\{d_Z, d_i\}$, and the frequencies are updated. Now a recursive call is made with new $\mathcal{Z} = \{d_Z, d_i\}$.

**Steps 9-10** Else if, $x_i$ available in $\mathcal{Z}$ is greater than the required cycles at $\{d_Z, d_i\}$, then we transfer to $\{d_Z, d_i\}$ as much of $x_i$ required, and the frequencies are updated to $f_{opt}$.

**Step 11** The minimum allowable frequency of each interval in $\{r_Z, d_i\}$ is updated. This can be done by considering only those jobs that have deadline at or before $d_i$. If the frequency thus obtained is greater than the current minimum allowable frequency, then the frequency is updated.

**Step 12** Now a recursive call is made with new $\mathcal{Z} = \{r_Z, d_i\}$.

**Steps 13-14** The minima of each interval in $\mathcal{Z}$ is updated with the present frequency of the intervals.

4. Proof of Correctness

**Theorem 3** Given, a set of independent, preemptive jobs with arbitrary release time, arbitrary deadline, and arbitrary execution time, the Optimal Energy Scheduler algorithm schedules the jobs on a single processor in an energy optimal manner.

*Proof:* The problem of optimizing a set of $n$ jobs with arbitrary release time, arbitrary deadline, and arbitrary execution time is approached using DP. DP produces the cheapest path for a given set. Since, in our algorithm, we do not produce all the frequencies possible by the jobs in job set, $\mathcal{J}$, we need to prove that the set of frequencies provided to the DP by our algorithm is a super set of the set of optimal frequencies. Since the algorithm is based on Theorem 2, it is sufficient to prove the correctness of the theorem.
It can be seen that Corollary 2 is false when either of Lemmas 5, 6 or 7 is false. This proves the necessary condition.

To prove the sufficiency condition, we show that Corollary 2 is true when all of Lemmas 5, 6 and 7 are true.

Assume that a feasible schedule $S$ satisfies the criteria in Lemmas 5, 6 and 7. Moreover, assume that $S$ is not energy-optimal. Therefore, a feasible schedule $S'$ must exist that has lower energy consumption than $S$.

Since $S'$ is an energy-optimal schedule, there should exist an interval $I'$ in $S'$ such that the frequency allocation $f'$ of $I'$ is lower than the corresponding interval $I$ in $S$. Two cases arise in the way jobs are scheduled in $I$.

*There exists some job $J$ scheduled in $I$ that can be feasibly re-scheduled in some other intervals of $S$ such that the energy consumption decreases:* In this case, $S$ does not satisfy Lemma 7.

*No job scheduled in $I$ can be re-scheduled in any other intervals of $S$ (with/without increasing energy consumption):* If there exists no job in $I$ that can be re-scheduled to any other intervals of $S$ even without increasing energy consumption of $S$ then, either intervals $I$ and $I'$ should have identical frequency allocation to obtain a feasible schedule, or Lemmas 5 or 6 has to be broken. If there exists a job in $I$ that can be re-scheduled to any other interval $I''$ (with higher frequency allocation) of $S$ by increasing energy consumption then; a job in $I''$ should be re-scheduled to $I$ (if possible) to satisfy Lemma 5 or job $J$ needs a frequency allocation at least equal to $f$ to generate a feasible schedule.

The algorithm requires $n$ stages to produce an optimal schedule for a set $J$ of size $n$. Each stage has several states associated with it. At each state a frequency is selected based on Theorem 2. By the end of each stage, the algorithm would have produced all the possible frequencies for jobs released at or after that stage, based
on Theorem 2. Since at each state of a stage, the algorithm tries to apply Lemma 2, the set of frequencies obtained as a result is a superset of the optimal frequencies possible till that stage.

During each stage of DP, no job $J_i$ in $J$ is scheduled beyond its deadline. Thus the schedule generated by the algorithm is feasible if a feasible schedule exists.

5. Performance

The performance of the algorithm can be evaluated as follows. Each of the steps of Algorithm 5 takes the following time complexity.

**Step 1** Sorting can be done in $O(n\log(n))$.

**Step 2-5** In the worst case these steps will take $O(n)$.

Each of the steps in Algorithm 6 takes the following time complexity.

**Step 1** The function is called $O(n)$ by Algorithm (5). In the worst case, each interval will be considered $O(n)$ times during each call. There can be a maximum of $n$ calls to the function. Therefore in the worst case it will take $O(n^2)$.

**Step 2-12** Each of the steps will execute to a maximum of $O(n)$.

**Step 13-14** Since the number of intervals possible is bounded by $2n - 1$, the worst case execution time of this step is bounded by $O(n)$.

Thus, the time complexity of the algorithm is $O(n^2)$. 
CHAPTER VII

RESULTS

In this chapter we evaluate the performance of both the processor models presented in Chapters V and VI using various task sets. Simulation experiments were conducted to calculate the effectiveness of the proposed algorithms in saving energy.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Voltage</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmeta Crusoe</td>
<td>1.3</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td>1.2</td>
<td>667</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>533</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>300</td>
</tr>
<tr>
<td>AMD K-6 IIIE</td>
<td>1.8</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>1.7</td>
<td>450</td>
</tr>
<tr>
<td></td>
<td>1.6</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>350</td>
</tr>
<tr>
<td></td>
<td>1.4</td>
<td>300</td>
</tr>
</tbody>
</table>

For brevity we refer to the algorithm described in Chapter V as *discrete* and Chapter VI as *continuous*. We use the term *base* to represent a *single-frequency* processor that executes at highest available frequency.

In this section we first compare the energy consumption of *discrete* to that of *base*. We then compare *discrete* to generalized network flow (GNF) model described
in [17]. We also compare the energy consumptions of discrete, continuous and base using task sets generated randomly as described later. We make a fair comparison of the models by alleviating any benefits of task timing by using a spectrum of utilization factor values. Both, GNF and discrete were solved using an ORMM solver [24] on a Pentium 4, 1.4 GHz machine with 512MB RAM.

The comparison of discrete with GNF and base is shown in Table 2. The processor characteristics used for comparison is shown in Table 1. The task set used in the above comparison was adopted from [17]. As can be seen from Table 2, both discrete and GNF outperform base for any value of processor utilization. Also it can be seen from Table 2 that discrete outperforms GNF in more than a couple of tasks. This is due to the fact that GNF is restricted to inter-task scheduling, while no such restriction applies to discrete. Thus in cases where it is optimal to run a task or interval at two frequencies, GNF is restricted to run at the higher of the two frequencies. Table 3 shows the percentage variation of energy consumption between discrete, GNF and base.

Time and space complexity to run both the algorithms are shown in Table 4. As can be seen from the table, the number of nodes and arcs required by GNF increases exponentially as the number of jobs increases, while that of discrete increases polynomially. The time required to execute GNF runs into minutes and in some cases into hours, while that of discrete remains less that two seconds in all the cases. Thus, discrete generates a schedule with lower energy consumption using a compact representation and polynomial time algorithm.

It can thus be seen that GNF, which is NP Complete is not scalable for higher number of frequencies. It also requires large number of arcs and nodes for representation and is computationally inefficient even for small number of problems.
<table>
<thead>
<tr>
<th>Model</th>
<th>Task Set</th>
<th>Processor Utilization</th>
<th>Energy consumption</th>
<th>Discrete</th>
<th>GNF</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Discrete</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GNF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Base</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>0.27</td>
<td>7452</td>
<td>7452</td>
<td>15548</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>0.43</td>
<td>17406</td>
<td>18976</td>
<td>35152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>0.54</td>
<td>16800</td>
<td>16800</td>
<td>28392</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>0.59</td>
<td>39104</td>
<td>39104</td>
<td>58136</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>0.65</td>
<td>52736</td>
<td>52736</td>
<td>74360</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>0.72</td>
<td>138751</td>
<td>138816</td>
<td>178464</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>0.75</td>
<td>61742</td>
<td>61760</td>
<td>81120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td>0.85</td>
<td>168831</td>
<td>169024</td>
<td>200096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNC</td>
<td>0.48</td>
<td>49528</td>
<td>51363</td>
<td>103336</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>0.27</td>
<td>11270</td>
<td>11270</td>
<td>18630</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>0.43</td>
<td>25480</td>
<td>25480</td>
<td>42120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>0.54</td>
<td>20580</td>
<td>20580</td>
<td>34020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>0.59</td>
<td>42140</td>
<td>42140</td>
<td>69660</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>0.65</td>
<td>58569</td>
<td>58573</td>
<td>89100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>0.72</td>
<td>155940</td>
<td>155992</td>
<td>213840</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>0.75</td>
<td>68988</td>
<td>69024</td>
<td>97200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td>0.85</td>
<td>195380</td>
<td>195333</td>
<td>239760</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNC</td>
<td>0.48</td>
<td>77707</td>
<td>89183</td>
<td>128455</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TABLE 3. Percentage variation of energy consumption in discrete, GNF and base

<table>
<thead>
<tr>
<th>Model</th>
<th>Task Set</th>
<th>(\frac{\text{Base} - \text{Discrete}}{\text{Base}})</th>
<th>(\frac{\text{GNF} - \text{Discrete}}{\text{GNF}})</th>
<th>(\frac{\text{Base} - \text{GNF}}{\text{Base}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crusoe</td>
<td>T1</td>
<td>52%</td>
<td>0%</td>
<td>52%</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>50%</td>
<td>8%</td>
<td>46%</td>
</tr>
<tr>
<td></td>
<td>T3</td>
<td>41%</td>
<td>0%</td>
<td>41%</td>
</tr>
<tr>
<td></td>
<td>T4</td>
<td>33%</td>
<td>0%</td>
<td>33%</td>
</tr>
<tr>
<td></td>
<td>T5</td>
<td>29%</td>
<td>0%</td>
<td>29%</td>
</tr>
<tr>
<td></td>
<td>T6</td>
<td>22%</td>
<td>0%</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td>T7</td>
<td>24%</td>
<td>0%</td>
<td>24%</td>
</tr>
<tr>
<td></td>
<td>T8</td>
<td>16%</td>
<td>0%</td>
<td>16%</td>
</tr>
<tr>
<td></td>
<td>CNC</td>
<td>54%</td>
<td>4%</td>
<td>52%</td>
</tr>
<tr>
<td>AMD</td>
<td>T1</td>
<td>40%</td>
<td>0%</td>
<td>40%</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>40%</td>
<td>0%</td>
<td>40%</td>
</tr>
<tr>
<td></td>
<td>T3</td>
<td>40%</td>
<td>0%</td>
<td>40%</td>
</tr>
<tr>
<td></td>
<td>T4</td>
<td>40%</td>
<td>0%</td>
<td>40%</td>
</tr>
<tr>
<td></td>
<td>T5</td>
<td>34%</td>
<td>0%</td>
<td>34%</td>
</tr>
<tr>
<td></td>
<td>T6</td>
<td>27%</td>
<td>0%</td>
<td>27%</td>
</tr>
<tr>
<td></td>
<td>T7</td>
<td>29%</td>
<td>0%</td>
<td>29%</td>
</tr>
<tr>
<td></td>
<td>T8</td>
<td>19%</td>
<td>0%</td>
<td>19%</td>
</tr>
<tr>
<td></td>
<td>CNC</td>
<td>40%</td>
<td>13%</td>
<td>31%</td>
</tr>
<tr>
<td>Task Set</td>
<td>Discrete</td>
<td>GNF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No. of Arcs</td>
<td>No. of Nodes</td>
<td>Time [mm:ss]</td>
<td>No. of Arcs</td>
</tr>
<tr>
<td>1</td>
<td>66</td>
<td>24</td>
<td>0.01</td>
<td>164</td>
</tr>
<tr>
<td>2</td>
<td>92</td>
<td>28</td>
<td>0.01</td>
<td>248</td>
</tr>
<tr>
<td>3</td>
<td>65</td>
<td>23</td>
<td>0.01</td>
<td>158</td>
</tr>
<tr>
<td>4</td>
<td>94</td>
<td>30</td>
<td>0.01</td>
<td>260</td>
</tr>
<tr>
<td>5</td>
<td>140</td>
<td>44</td>
<td>0.01</td>
<td>404</td>
</tr>
<tr>
<td>6</td>
<td>378</td>
<td>90</td>
<td>0.01</td>
<td>1140</td>
</tr>
<tr>
<td>7</td>
<td>121</td>
<td>37</td>
<td>0.01</td>
<td>302</td>
</tr>
<tr>
<td>8</td>
<td>328</td>
<td>79</td>
<td>0.01</td>
<td>979</td>
</tr>
<tr>
<td>CNC</td>
<td>896</td>
<td>213</td>
<td>0.01</td>
<td>2138</td>
</tr>
</tbody>
</table>

TABLE 5. Energy consumption of discrete and continuous on 5-\( \mathcal{T} \) set

<table>
<thead>
<tr>
<th>Task set</th>
<th>Energy consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>continuous</td>
<td>discrete</td>
</tr>
<tr>
<td>5-( \mathcal{T} )1</td>
<td>64619</td>
</tr>
<tr>
<td>5-( \mathcal{T} )2</td>
<td>186298</td>
</tr>
<tr>
<td>5-( \mathcal{T} )3</td>
<td>380464</td>
</tr>
<tr>
<td>5-( \mathcal{T} )4</td>
<td>663095</td>
</tr>
</tbody>
</table>
TABLE 6. Energy consumption of *discrete* and *continuous* on 20-\(\mathcal{T}\) set

<table>
<thead>
<tr>
<th>Task set</th>
<th>Energy consumption</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>continuous</td>
<td>discrete</td>
</tr>
<tr>
<td>10-(\mathcal{T}1)</td>
<td>179485</td>
<td>352040</td>
</tr>
<tr>
<td>10-(\mathcal{T}2)</td>
<td>517442</td>
<td>704816</td>
</tr>
<tr>
<td>10-(\mathcal{T}3)</td>
<td>1057290</td>
<td>1057802</td>
</tr>
<tr>
<td>10-(\mathcal{T}4)</td>
<td>1841150</td>
<td>1841303</td>
</tr>
</tbody>
</table>

TABLE 7. Energy consumption of *discrete* and *continuous* on 10-\(\mathcal{T}\) set.

<table>
<thead>
<tr>
<th>Task set</th>
<th>Energy consumption</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>continuous</td>
<td>discrete</td>
</tr>
<tr>
<td>20-(\mathcal{T}1)</td>
<td>29737</td>
<td>58432</td>
</tr>
<tr>
<td>20-(\mathcal{T}2)</td>
<td>85500</td>
<td>116863</td>
</tr>
<tr>
<td>20-(\mathcal{T}3)</td>
<td>174354</td>
<td>175295</td>
</tr>
<tr>
<td>20-(\mathcal{T}4)</td>
<td>306013</td>
<td>306103</td>
</tr>
</tbody>
</table>
Above, we saw that discrete outperforms both GNF as well as base in generating a schedule that consumes lower energy. In the following, we simulate a continuous-frequency processor from AMD K-6 processor shown in Table 1. The frequency to voltage relation is calculated from the linear relation of the voltage and frequency as shown in the Table 1. We assume that the continuous-frequency processor can run at any frequency lower than the maximum frequency available for the AMD processor (500 MHz).

The task sets used in the following is generated randomly. Similar methods of generating workload is already used in [19]. We generate three different kinds of task sets with four processor utilizations. A mixed class task set is created by randomly generating task periods in the range of \((1 - 1000)\)\(\mu\)seconds, by dividing the range into short \((0 - 10)\)\(\mu\)seconds, medium \((11 - 100)\)\(\mu\)seconds, and large \((101 - 999)\)\(\mu\)seconds intervals. Two random tasks are generated in the short and large intervals while only one task is generated in the medium period range. We then simulate a homogenous workload with a shorter period range by generating tasks randomly in the period range of \((0 - 25)\)\(\mu\)seconds and \((25 - 50)\)\(\mu\)seconds. The results of simulating these workloads are shown in Tables 5, 7, and 6. The percentage of variation is shown in Table 8.

It can be seen from the tables that frequency generated by discrete is almost equal to that of continuous as the processor utilization goes up (above 60%). When utilization is low (below 40%), continuous generates a schedule that consumes lower energy than discrete. This is because discrete cannot go below a certain minimum frequency (300 MHz in this case) assigned to it.
TABLE 8. Variation of energy consumption of discrete, continuous and base

<table>
<thead>
<tr>
<th>Processor Utilization</th>
<th>5-Τ Set</th>
<th>10-Τ Set</th>
<th>20-Τ Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>20%</td>
<td>49%</td>
<td>49%</td>
<td>49%</td>
</tr>
<tr>
<td>40%</td>
<td>27%</td>
<td>27%</td>
<td>27%</td>
</tr>
<tr>
<td>60%</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
</tr>
<tr>
<td>80%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>
CHAPTER VIII

CONCLUSIONS AND FUTURE DIRECTIONS

In this thesis, we developed two new algorithms to solve the problem of optimal energy scheduling for hard real-time systems.

First, we developed an algorithm to schedule a set of jobs with arbitrary release time, arbitrary deadline and arbitrary execution time on a processor with given set of discrete frequencies. We use the network flow graph to solve the problem. The algorithm utilizes intra-task scheduling model, thus obtaining the minimum energy consumption possible on a given set of frequency.

Second, we developed an algorithm to schedule a set of jobs with arbitrary release time, arbitrary deadline and arbitrary execution time, on a continuous frequency processor such that the energy consumed is minimum.

We first solve a simpler version of the problem by assuming identical release time for all the jobs. We developed a polynomial time algorithm to solve the same. Later, we propose a polynomial time, dynamic programming based approach to solve the optimal energy scheduling problem.

We assume a non-linear relation of voltage to energy. Though, we assume that energy is proportional to square of voltage, our model is generalized and would work under any model with a non-linear relation of energy to voltage. We provide proofs for our claims of optimality and results to substantiate our claims. It is our belief that the results obtained are the best so far in the literature.

The continuous frequency allocation assumes that the available number of frequencies is not limited. This is acceptable in some systems where resource bandwidth is truly variable on a continuous scale. In many other cases, such as in
the design of *application-specific* processors, the number of available frequencies is bounded. For such cases, algorithms must be found that generate frequency level bounded energy-optimal schedules for *continuous-frequency* processors.
REFERENCES


VITA

John V. George received his B. Tech degree in Computer Science and Engineering from Mahatma Gandhi University, Kerala, India in 2001 and his M.S. degree from Texas A&M University, College Station, Texas in 2005. His major interests are Real-Time Scheduling and Embedded Systems. While pursuing his masters, he also worked as a research assistant in the area of embedded systems. He can be reached at:

Edakkattukudy House
Kozhipilly P.O Kothamangalam
Ernakulam Kerala India 686691.

This thesis was typed by John V. George.