VOLTAGE SCHEDULING TECHNIQUES FOR DYNAMIC VOLTAGE
SCALING PROCESSORS WITH PRACTICAL LIMITATIONS

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

by

Bren Christopher Mochocki, B.S.

Xiaobo (Sharon) Hu, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
March 2004
Abstract

by

Bren Christopher Mochocki

Voltage scheduling is an essential technique used to exploit the benefit of dynamic voltage scaling processors. Though extensive research exists in this area, processor limitations such as transition overhead and voltage level discretization are not included simultaneously in any research processor model to date. Algorithms that account for individual limitations are not sufficient because some complications only emerge when limitations are considered simultaneously. We present two algorithms that yield valid results given an arbitrarily large transition time overhead. The first, LPEDF, offers a simple implementation, while the second, UAEDF, accounts for all three processor limitations to further reduce energy consumption.
DEDICATION

I dedicate this work to three important people in my life. To my father, Henry Mochocki, who’s creativity and innovation have always been an inspiration, my mother, Christine Mochocki, who’s selfless love has carried me through countless rough times, and Gregory Burkett, who’s confidence in me never failed even when my own proved weak. These qualities are not mutually exclusive, however, which is why these people are so special.
CONTENTS

FIGURES ......................................................... v

TABLES ....................................................... vi

SYMBOLS ...................................................... vii

PREFACE ......................................................... ix

CHAPTER 1: INTRODUCTION ............................... 1
  1.1 Overview .............................................. 1
  1.2 Practical Limitations on DVS Processors .......... 2
  1.3 Problem of Interest ................................. 4
  1.4 Previous Work ....................................... 5
  1.5 Contributions ...................................... 6
  1.6 Thesis Outline ..................................... 7

CHAPTER 2: PRELIMINARIES ............................... 8
  2.1 System Model ......................................... 8
  2.2 Low-Power Earliest Deadline First (LPEDF) ....... 10
  2.3 Motivational Example ............................. 12

CHAPTER 3: A BASIC ALGORITHM FOR TIME OVERHEAD ... 16

CHAPTER 4: A UNIFIED ALGORITHM FOR NON-IDEAL DVS PROCESSORS ........................................ 23
  4.1 Improving the Basic Algorithm .................... 23
  4.2 Discrete Voltage/Frequency Levels ................ 27
  4.3 Transition Energy Overhead ...................... 29
  4.4 The Unified Algorithm ............................ 29
FIGURES

2.1 The process used by LPEDF to find the optimal voltage schedule on an ideal DVS processor. ................................. 11

2.2 (a) An example set of jobs. (b) The optimal voltage schedule using LPEDF. (c) The LPEDF schedule, modified by inserting $\Delta t = 100$ at each transition, and then rescaling $T_2$, $T_3$ and $T_4$ to maintain the same executed workload. (d) The schedule produced by M-LPEDF. . 13

3.1 The process used by TOEDF to find a valid voltage schedule. . . . . 19

3.2 The job set from Figure 2.2(a), scheduled by TOEDF. ............... 21

4.1 Flow diagram of UAEDF. ........................................ 30

5.1 A plot of the 5 valid voltage vs. frequency operation points of the 1000 MHz AMD Mobile Athlon4 processor from Table 5.1. The points are interpolated using a second order polynomial. ............... 38

5.2 Average increase in energy above LPEDF for 100 random job sets. . 42

5.3 Increase in energy above LPEDF for the CNC job set. ............... 43

5.4 Increase in energy above LPEDF for the Avionics job set. ........... 44

5.5 Increase in energy above LPEDF for the Video Phone job set. ....... 45
TABLES

5.1 PERFORMANCE SETTINGS OF AMD’S MOBILE ATHLON 4 . 35
5.2 CORE VOLTAGE/CURRENT ON AMD’S MOBILE ATHLON 4 . 36
5.3 POWER CHARACTERISTICS OF AMD’S MOBILE ATHLON 4 . 37
A.1 STATISTICS FOR THE RANDOMLY GENERATED JOB SETS . 50
A.2 THE COMPUTERIZED NUMERIC CONTROLLER TASK SET . 50
A.3 DEADLINE MODIFIED AUDIO AND VIDEO PHONE TASK SET 50
A.4 DEADLINE/PHASE SHIFT MODIFIED AVIONICS TASK SET . 51
SYMBOLS

\( P_{\text{total}} \)  Total Power
\( P_{\text{dynamic}} \)  Dynamic Power
\( P_{\text{static}} \)  Static or leakage power
\( P_{\text{direct}} \)  Direct path power
\( V_{\text{DD}} \)  Supply voltage
\( V_T \)  Threshold voltage
\( f_{\text{clk}} \)  Clock frequency
\( C_{\text{SW}} \)  Average switch capacitance
\( \alpha \)  Velocity saturation of the majority carrier
\( I_{\text{static}} \)  Static or leakage current
\( I_{\text{direct}} \)  Direct-path current
\(|\mathcal{A}|\)  Number of elements in the set \( \mathcal{A} \)
\( \mathcal{J} \)  A set of independent jobs, sorted by Earliest Deadline First
\( t_{\text{LS}} \)  The latest start time of job set \( \mathcal{J} \)
\( J_i \)  An individual job in set \( \mathcal{J} \)
\( t_{\text{LS}}(i) \)  The latest start time of job \( J_i \)
\( r_i \)  The release time of job \( J_i \)
\( d_i \)  The absolute deadline of job \( J_i \)
\( c_i \)  The worst case execution cycles of job \( J_i \)
\( WCET_i \)  The worst case execution time of job \( J_i \)
\( \mathcal{S} \)  A set of valid processor speeds/frequencies, sorted in increasing order
\( S_i \)  A speed in the set \( S \)

\( S_{\text{max}} \)  The maximum speed in set \( S \). (i.e. \( S_{|S|} \))

\( s^* \)  A single constant speed

\( T \)  A set of constant voltage/frequency intervals

\( T_i \)  An interval in the set \( T \)

\( J_i \)  A set of jobs scheduled for execution in the interval \( T_i \)

\( t_s(i) \)  The start time of interval \( T_i \)

\( t_f(i) \)  The finish time of interval \( T_i \)

\( |T_i| \)  The length of interval \( T_i \), i.e., \( |T_i| = t_f(i) - t_s(i) \)

\( s_i \)  The speed of interval \( T_i \)

\( \Delta t \)  A processor’s time transition overhead

\( \Delta E \)  A processor’s energy transition overhead

\( t \)  An instant in time

\( i \)  A set index

\( j \)  A set index

\( k \)  A set index

\( i_{\text{prev}} \)  A set index used in some previous loop iteration of \( \text{UAEDF} \)

\( T_{\text{adj}} \)  The set of intervals adjacent to some interval \( T_k \in T \), where \( T_{\text{adj}} \subset T \).
PREFACE

The research material presented in this thesis represents the result of a joint effort with my advisor, Dr. Xiaobo Sharon Hu, and our collaborator from the University of South Carolina, Dr. Gang Quan. This research was funded in part by the National Science Foundation under grant numbers MIP-9701416, CCR-9988468 and CCR02-08992, and By the Arthur J Schmitt Foundation’s graduate research fellowship.

A preliminary version of this work is published by the ACM press in the proceedings of the 2002 International Conference on Computer Aided Design (ICCAD) [26]. A condensed version is currently under review by IEEE for their Transactions on Computer Aided Design (TCAD).
CHAPTER 1

INTRODUCTION

1.1 Overview

The demand for mobile and pervasive computing devices has made low power computing a critical technology. One of the most effective ways to reduce energy consumption in Complementary Metal-Oxide-Semiconductor (CMOS) processors is called Dynamic Voltage Scaling (DVS), i.e. dynamically varying a processor’s supply voltage ($V_{DD}$) and clock frequency ($f_{clk}$) simultaneously.

The power consumption in CMOS is a combination of three main components, the dynamic, static or leakage, and direct path power. The expressions commonly used to represent this power consumption are given in equations 1.1 through 1.4, along with equation 1.5 which describes the propagation delay and equations 1.6 and 1.7 that describe CMOS energy consumption.

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}} + P_{\text{direct}}$$  \hspace{1cm} (1.1)

$$P_{\text{dynamic}} = C_{SW}V_{DD}^2f_{clk}$$  \hspace{1cm} (1.2)

$$P_{\text{static}} = I_{\text{leak}}V_{DD}$$  \hspace{1cm} (1.3)

$$P_{\text{direct}} = I_{\text{direct}}V_{DD}$$  \hspace{1cm} (1.4)

$$t_p = \frac{V_{DD}}{(V_{DD} - V_T)^\alpha}, \quad \alpha = [1, 2]$$  \hspace{1cm} (1.5)

$$E = \int Pdt$$  \hspace{1cm} (1.6)
\[ E_{\text{dynamic}}([t_1, t_2]) = C_{SW}V_D^2f_{\text{clk}}(t_2 - t_1) \] (1.7)

In these equations, \( C_{SW} \) is the average switching capacitance (1.1 and 1.7), \( I_{\text{leak}} \) and \( I_{\text{direct}} \) are the leakage and direct-path currents respectively (1.3 and 1.4) and \( t_p \) represents the propagation delay (1.5). The constant \( \alpha \) is the velocity saturation of the majority carrier (1.5), and is typically around 1.3 for advanced CMOS circuits. Dynamic power is consumed whenever a switch takes place, static power consumption occurs when a CMOS device is idle and the leakage current is non-zero, and direct path power consumption occurs because a low-high or high-low voltage transition cannot take place instantaneously, and for a short time there is a direct path from the supply voltage to ground [31, 8].

Though all three sources of power include the \( V_{DD} \) term and can thus benefit from DVS, the most critical term, \( P_{\text{dynamic}} \), also has the largest dependency on \( V_{DD} \). In fact, because \( f_{\text{clk}} \) is always scaled in proportion to \( V_{DD} \), the dynamic power has a cubic dependence on the supply voltage. Energy is the sum of the power consumption over time (1.6), and the dynamic energy consumption for a constant voltage interval over the range \([t_1, t_2]\) is given in equation 1.7. The clock speed of the processor, \( f_{\text{clk}} \), only has a linear dependence on \( V_{DD} \) when the threshold voltage is much larger than the supply voltage, i.e., \( V_T >> V_{DD} \). Therefore, a small decrease in the operation speed will cause a proportional change in execution time, but causes a large decrease in both power and energy consumption [7]. The \( V_T >> V_{DD} \) assumption holds for current and near-future CMOS technology, according to the 2003 technology road map for semiconductors [35].

1.2 Practical Limitations on DVS Processors

Various DVS processors are commercially available, including Intel’s XScale [15], AMD’s Mobile Athlon [3], and Transmeta’s Crusoe processor [11]. Several research
groups have also developed their own variable voltage systems. Burd and Brodersen implemented a variable voltage system using the ARM8 core [6], while Pouwelse, Langendoen and Sips constructed a similar system using the SA-1100 [29]. The Compaq iPAQ implements frequency scaling without DVS, but future versions are likely to have DVS also [10].

A common problem with all of these systems is that they suffer from practical limitations of DVS, including transition overhead and voltage level discretization. Transition overhead is the time and energy consumed during a frequency/voltage switch. The time overhead can be on the order of tens of microseconds [6, 15], hundreds of microseconds [3, 29, 11], or milliseconds [10]. The microsecond time overheads result from the DC-DC converter changing $V_{DD}$ or the phase-locked loop changing $f_{clk}$, while the larger millisecond overheads are sometimes required to synchronize the CPU with off chip components such memory. Energy overhead has two sources, the CPU and the DC-DC converter. Most DVS processors block instructions during a transition, so the CPU component of the energy overhead is the processors idle power times the length of the transition interval. The DC-DC converter’s energy consumption is given in equation 1.8 and is discussed in detail in [5].

$$\Delta E_{DC} = \eta C_{DD} |V_{DD1} - V_{DD2}|$$  \hspace{1cm} (1.8)

The converter’s load capacitance is given by $C_{DD}$, and is around $100 \mu F$ for a typical low power converter. The efficiency is given by $\eta$ and is typically around 0.9. The source and target voltage levels are given by $V_{DD1}$ and $V_{DD2}$.

The third practical limitation is voltage level discretization. Theoretical research on DVS systems tends to assume a continuously variable DVS processor, while typical processors can only use between 2 and 14 different voltage levels. Voltage-scheduling algorithms that don’t account for this limitation are forced to round up
their scheduled voltages to the next available level to guarantee system deadlines. This method is not efficient from an energy standpoint, as we show later.

1.3 Problem of Interest

To maximally exploit the benefits of a DVS processor, voltage scheduling, the selection of voltage levels and operating frequencies, is indispensable. A large number of research results have been published on voltage scheduling for DVS processors. These results differ in many aspects, such as the type of applications (e.g., real-time or non real-time) considered, the type of systems (e.g., single or multiple processors) used, the location (intra-task v.s. inter-task) where a voltage change is allowed, the execution style (e.g., on-line or off-line) of the voltage scheduling algorithms, etc.

It is not difficult to see that non-ideal properties of DVS processors, such as discrete voltage levels and transition overhead, can effect voltage scheduling results and deserve careful study.

In this paper, we focus on voltage scheduling for a set of real-time jobs executed by a single DVS processor. Many embedded applications can be described by such a model. In particular, we study off-line, inter-job voltage scheduling where the real-time jobs are executed according to the preemptive Earliest Deadline First (EDF) scheduling scheme [22]. Preemptive EDF is an optimal scheduling algorithm and has been adopted by many real-time systems [23]. Inter-job (or inter-task) voltage scheduling is realized by the operation system, which is less intrusive and more portable for a given application. Off-line scheduling does not compete for resources with the actual application and hence can afford to use more sophisticated algorithms. Though off-line scheduling cannot handle dynamic variations, it can often be used as a complement to on-line approaches. The uniqueness of our work is that the algorithms developed here account for practical limitations of DVS processors,
including transition time and energy overhead as well as discrete voltage levels.

1.4 Previous Work

Substantial research exists for scheduling real-time applications on DVS processors, e.g., [18, 28, 27, 30, 34, 33, 36, 37, 4, 12], some of which have considered off-line, inter-job voltage scheduling for preemptive EDF based real-time systems, e.g., [36, 4, 12]. However, a limited amount of work has examined voltage scheduling in the presence of practical limitations. Some of these consider only discrete voltage levels. For example, Chandrasena et al. [9] introduce a rate selection algorithm for a DVS processor with limited voltage levels, but the algorithm provides no deadline guarantee for the tasks. In [20] and [21], Lee et al. consider discrete voltage levels in dynamic (i.e., on-line) voltage scheduling for periodic tasks. Their method, called time slicing, requires that tasks be divided into subtasks (or slots), which is not always possible. Even if the division is possible, preemption is not allowed within a sub-task, so this method cannot be applied to the preemptive scheduling problem we address here. Kwon et al. give an optimal intra-task scheduling algorithm under the preemptive EDF scheme to match a discrete set of voltage levels [19]. Saewong and Rajkumar give an in-depth analysis of the ideal placement of a set of discrete voltage levels for a DVS processor, and conclude that the energy increase due to voltage/frequency quantization is inversely proportional to the number of levels [32]. Since the above approaches ignore transition overhead, they tend to introduce more transitions in order to better match discrete voltage levels, which further exasperate the impact of transition overhead.

A number of researchers have studied voltage scheduling when transition overhead is not negligible. Manzak et al. [25] address the transition time overhead by linearly increasing the total required execution time or decreasing the processor
utilization. Such adjustments may lead to either a deadline miss or an overly pessimistic design. In [13], Hong et al. present a heuristic algorithm that accounts for transition overhead during static scheduling, but assumes both the availability of any voltage level and continuous execution of instructions during a transition. Neither assumption is true for most DVS systems [3, 11, 15, 29]. AbouGhazaleh et al. propose an intra-task voltage scheduling method that accounts for transition overhead. Their method, however, requires both compiler support and that the source code be engineered to give voltage selection “hints” to the operating system [1, 2]. Hsu and Kremer also present a compiler driven DVS algorithm, but hard deadlines are not guaranteed [14]. Saewong and Rajkumar account for large time overheads using a method called **Sys-Clock**, that selects the minimum constant speed that will finish all jobs by their deadlines; essentially avoiding transition overhead by having no transitions at all [32].

1.5 Contributions

In this paper, we present several observations regarding the impact of transition overhead on executing real-time jobs according to the preemptive EDF scheme. These observations show that transition time overhead can cause deadline misses as well as a significant increase in energy consumption if not handled carefully. A basic algorithm is devised to guarantee that no deadline violations occur in the presence of transition time overhead. Building on the basic algorithm, we developed an algorithm that considers both transition time and energy overheads as well as discrete voltage levels. A large number of experiments have been conducted for both real-world and randomly generated examples to demonstrate the effectiveness of our algorithms. Careful analysis of the experimental results helps us to draw several conclusions regarding the non-ideal properties of DVS processors and the
performance of our proposed algorithms.

1.6 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 summarizes the relevant background material, including system models, motivation examples, and so forth. Chapter 3 describes a basic algorithm that handles time transition overhead. Chapter 4 improves the basic algorithm in terms of energy savings and accounts for both energy overhead and discrete voltage levels, in addition to time overhead. Chapter 5 presents some experimental results. Chapter 6 concludes the thesis and offers several directions for future work.
2.1 System Model

We consider real-time applications consisting of a set of independent jobs, $\mathcal{J} = \{J_1, ..., J_n\}$ with each job $J_i \in \mathcal{J}$ having a release time $r_i$, a deadline $d_i$, and worst case execution cycles $c_i$. The job set is to be executed on a DVS processor whose power consumption is a convex function of the processor speed (frequency) [8]. The convexity assumption holds so long as the switching power is one of the main contributors of the total power, which is the case for current and near future CMOS devices [35].

The DVS processor can operate at a finite set of supply voltage levels $\mathcal{V} = \{V_1, ..., V_{\text{max}}\}$, each with an associated speed. To simplify the discussion, we normalize the processor speeds by $S_{\text{max}}$, the speed corresponding to $V_{\text{max}}$, and thus we have $\mathcal{S} = \{S_1, ..., 1\}$.

Changing from one voltage level to another takes a fixed amount of time, referred to as the transition interval (denoted as $\Delta t$), and consumes a variable amount of transition energy (denoted as $\Delta E$).
The transition interval length for a DVS processor alone is usually on the order of 10 to 100 \( \mu s \) [3, 6, 29]. However, when considering synchronizing with other components in a system, the length can be on the order of milliseconds [32, 10]. Transition energy includes both the energy consumed by the DC/DC converter and the CPU. No instructions are executed during a transition.

The above DVS processor model captures the main features of most commercial DVS processors [3, 29]. A variable length transition interval (e.g. the one described in [6]) can be approximated by a fixed length interval equal to the maximum switching time. For processors that do not block instructions during a transition (e.g., [6]), a schedule that assumes blocking during a transition can be pessimistic, but it will guarantee a valid voltage schedule. As in most voltage scheduling work, we assume that each job consumes an equal amount of energy per cycle at a given speed, which is a valid assumption for many applications.

We introduce some notation below which will be used throughout the paper. We denote a time interval by \( T = [t_s, t_f] \) and the interval length by \( |T| = t_f - t_s \). The intensity, \( s(T) \), of an interval \( T \) is the minimum processor speed required to finish jobs released inside the interval on or before \( t_f \). It can be readily verified that

\[
s(T) = \frac{\sum_{J_k \in J} c_k}{|T|}, \quad r_k \geq t_s, \quad d_k \leq t_f \tag{2.1}
\]

For a given job set \( J \), the critical interval is defined as the interval with the highest speed. The algorithms we describe in this paper operate by iteratively identifying and scheduling each critical interval. The critical interval found at iteration \( i \) is denoted by \( T_i = [t_s(i), t_f(i)] \) and executes a set of jobs \( J_i \in J \) at a speed \( s_i \). Two critical intervals \( T_i \) and \( T_j \) with \( i > j \) are said to be adjacent if one of the following conditions is true: (1) \( t_f(j) < t_s(i) \) and there are no other critical intervals or unscheduled jobs in \([t_f(j), t_s(i)]\), i.e., \( T_j \) is to the left of \( T_i \), (2) \( t_f(i) < t_s(j) \) and
there are no other critical intervals or unscheduled jobs in \([t_f(i), t_s(j)], \) i.e., \(T_j\) is to the right of \(T_i\), or (3) \(t_s(i) < t_s(j)\) and \(t_f(j) < t_f(i)\), i.e., \(T_i\) contains \(T_j\). Also, for the rest of this paper we may omit the word \textit{transition} when referring to \textit{transition} time/energy overhead.

2.2 Low-Power Earliest Deadline First (LPEDF)

We briefly review the voltage scheduling algorithm \textbf{LPEDF}, presented in [36], as it is referenced throughout the paper. We will briefly review LPEDF in Algorithm 1 and Figure 2.1.

\begin{algorithm}
\caption{[\(T\)] = LPEDF (\(J\))}
\begin{algorithmic}[1]
\State \textbf{Input:} The job set \(J\).
\State \textbf{Output:} A valid voltage Schedule \(T = \{(T_i, s_i)\}\), consisting of a set of critical-interval/speed pairs.
\State \(i := 1\)/* the critical interval index */
\While {\(J\) is not empty}
\State Find the next critical interval \(T_i = [t_s(i), t_f(i)]\) with \(s_i = s(T_i)\);
\State Insert \((T_i, s_i)\) into \(T\);
\State Remove all jobs \(J_i\) from \(J\);
\For {every release time or deadline, \(t_{sch}\), where \(t_{sch} > t_s(i)\)}
\If {\(t_{sch} \leq t_f(i)\)}
\State \(t_{sch} := t_s(i)\);
\Else
\State \(t_{sch} := t_{sch} - |T_i|\);
\EndIf
\EndFor
\State \(i := i + 1\);
\EndWhile
\State \textbf{Return} \(T\);
\end{algorithmic}
\end{algorithm}

LPEDF finds off-line an optimal voltage schedule for a set of independent tasks executed according to the preemptive EDF policy. It assumes an ideal DVS processor without transition overhead. The general idea is to iteratively identify (line 5), schedule (lines 6 and 7) and remove (lines 8-14) each critical interval. Lines 8-14 essentially “squeeze” the critical interval to a single time point at \(t_s(i)\) by reducing
Figure 2.1. The process used by LPEDF to find the optimal voltage schedule on an ideal DVS processor.
all release times or deadlines inside $T_i$ to $t_s(i)$ and then reducing all release times or
deadlines after $t_f(i)$ by $|T_i|$. For the rest of the paper when we refer to squeezing or
compressing an interval, we mean performing a similar operation.

Algorithm 1 is greedy in the sense that it always picks the *critical interval* to
schedule first. Consequently, the intervals are identified according to a monotonically
non-increasing order of their associated speeds. Due to the convexity of the power
function, this monotonicity property, summarized formally in Lemma 1, is beneficial
when constructing voltage schedules. For the proof of Lemma 1 and more details
on LPEDF, we direct the readers to [36].

**Lemma 1** Critical intervals found by successive iterations of LPEDF
are monotonically non-increasing in intensity, that is, $s_i \geq s_j$ if $i \leq j$.

### 2.3 Motivational Example

To illustrate the impact of ignoring transition overhead, we present the following
example. Consider the job set in Figure 2.2(a), which contains four jobs (where $\Delta$
represents a job release time and $\triangledown$ a job deadline). The optimal voltage schedule
by LPEDF, assuming both $\Delta t$ and $\Delta E$ are zero, is given in Figure 2.2(b). Suppose
the same set of jobs is scheduled on a DVS processor with $\Delta t = 100$. (Note that in
the example job set the job active durations, i.e., $d_i - r_i$, are between 4 to 9 times
$\Delta t$. In general, job active durations can be any value.)

A straightforward approach to include time overhead is to (1) insert a transition
interval at each speed change and (2) rescale the speed of the interval with the
lower speed to ensure that the same amount of work is completed. (Increasing the
speed of the interval with the higher speed would lead to reduced energy savings due
to the convexity property of the power function.) The resulting schedule is shown
in Figure 2.2(c). The speed of $T'_2$, the modified interval $T_2$, was calculated using
equation 2.2.
Figure 2.2. (a) An example set of jobs. (b) The optimal voltage schedule using LPEDF. (c) The LPEDF schedule, modified by inserting $\Delta t = 100$ at each transition, and then rescaling $T_2$, $T_3$ and $T_4$ to maintain the same executed workload. (d) The schedule produced by M-LPEDF.
There are several problems with the schedule in Figure 2.2(c). First, $s'_2$ and $s'_3$ are now higher than $s_1$, i.e., the speed of a critical interval is higher than that of a critical interval identified in a previous iteration of LPEDF. We refer to this as a **monotonicity violation** because it violates Lemma 1. Note that $s'_2$ and $s'_3$ also surpass the normalized maximum of 1, referred to as **overshoot**. The presence of overshoot implies that the schedule is not valid. Second, $J_3$ will never be executed. Notice that $r_3 = 500$ and $d_3 = 1050$, but no cycles can be executed during the transition intervals $[500, 600]$ and $[1000, 1100]$ and the critical interval $[600, 1000]$ is completely utilized by $J_2$. We refer to this problem, where a job is not executed because it is contained in transition intervals, as an **execution violation**. Finally, notice that there is a sliver of idle time in the interval $[1192, 1200]$. When $T'_2$ was formed from $T_2$ to accommodate the transition interval, it still included the execution cycles of $J_3$, even though $T'_2$ is outside $[r_3, d_3]$. The extra time was used by $J_4$ during $T'_2$, so the idle part of $T'_3$ is not required to complete $J_4$. Because equation 2.2 fails to take into account release times and deadlines when rescaling the speed, it can result in schedules that require jobs to execute before they are released or after their deadlines.

A more sophisticated modification to LPEDF follows: Instead of compressing just the critical interval $T_i = [t_s(i), t_f(i)]$ down to a single time point, compress the interval $[t_s(i) - \Delta t, t_f(i) + \Delta t]$ and adjust adjacent jobs accordingly. We refer to this approach as **M-LPEDF**. With M-LPEDF, a job $J_i$ will never be scheduled for execution outside $[r_i, d_i]$. Applying M-LPEDF to the system in Figure 2.2(a), we obtain the schedule shown in Figure 2.2(d). Unfortunately, the schedule in Figure 2.2(d) still has monotonicity and execution violations since $s_2 > s_1$ and $J_3$
is contained in the transition intervals about $T_1$. Furthermore, the schedule is not still not feasible due to the overshoot of $T_2$.

From the above discussions, it is clear that accounting for time overhead is not just a simple process of locally adjusting the optimal LPEDF solution. Care must be taken during the scheduling process to ensure that the resulting schedule is valid. Considering energy overhead and discrete voltage levels adds further complications. In the following, we propose our approaches to solve this problem. To simplify our discussion, we will first assume that $\Delta E$ is negligible and that the processor’s voltage can vary continuously. We will remove these assumptions later.
CHAPTER 3

A BASIC ALGORITHM FOR TIME OVERHEAD

In this section, we present an algorithm that can eliminate both monotonicity and execution violations as well as overshoots, thus guaranteeing a feasible schedule in the presence of time overhead. Our algorithm is built on M-LPEDF.

Regarding monotonicity violations induced by M-LPEDF, we have observed that any critical interval that violates Lemma 1 must be adjacent to the critical interval identified in the previous iteration. (Note that generally critical intervals found in successive iterations are not necessarily adjacent to one another). This observation is stated formally in Lemma 2.

Lemma 2 Let $T_{i-1}$ and $T_i$ be two critical intervals obtained by M-LPEDF. If $s_{i-1} < s_i$, then $T_{i-1}$ is adjacent to $T_i$.

Proof: To schedule $T_{i-1} = [t_s(i-1), t_f(i-1)]$, M-LPEDF compresses $T'_{i-1} = [t_s(i-1) - \Delta t, t_f(i-1) + \Delta t]$. This modification does not change the workload distribution of jobs outside $T'_{i-1}$. Only intervals adjacent to or overlapping $T'_{i-1}$ are altered; shortened by up to $\Delta t$ time or by exactly $2\Delta t$ time if $T_i$ contains $T'_{i-1}$.

Thus, only intervals adjacent to $T_{i-1}$ may experience an increase in intensity in the next iteration. □

As implied in the proof for Lemma 2, a monotonicity violation occurs when the allotted time intervals for executing some jobs are shortened due to the addition of transition intervals. The result is that these jobs require a higher speed in order
to meet their deadlines. To remove such a violation, we could simply eliminate the
transitions by merging jobs in the violation interval with those in the previously
found critical interval. However, we need to be sure that such a merge will not lead
to any deadline misses. Lemma 3 below provides this guarantee.

Lemma 3 Let $T_{i-1}$ and $T_i$ be two critical intervals obtained by M-LPEDF
with $s_{i-1} < s_i$. The minimum speed at which every job $J_k \in (\mathcal{J}_{i-1} \cup \mathcal{J}_i)$
can execute and still meet its deadline is $s_{i-1}$.

Proof: According to Lemma 2, $T_{i-1}$ and $T_i$ must be adjacent. Therefore, if the speed
$s_{i-1}$ is applied to both of these intervals, no voltage transition occurs. According to
Lemma 1, $s_{i-1}$ is the minimum speed required to guarantee the deadlines of jobs in
$\mathcal{J}_{i-1}$ and is greater than the speed needed to guarantee the deadlines of jobs in $\mathcal{J}_i$
when no time overhead is present. Therefore, $s_{i-1}$ is the minimum speed required
for every job in $(\mathcal{J}_{i-1} \cup \mathcal{J}_i)$ to meet its deadline. □

Based on Lemmas 2 and 3, we can keep track of monotonicity violations and remove
them whenever they occur. In fact, these lemmas also help eliminate execution vi-
olutions. Observe that an execution violation is just a special case of a monotonicity
violation, where the speed required by the violation interval is $\infty$. For instance, in
Figure 2.2(d), after applying M-LPEDF at the first iteration, $T_1$ is identified and
$[600 - 100, 1000 + 100]$ is compressed. This results in the modification of $r_3$ and
d_3 where $r'_3 = d'_3 = 500$. During the next scheduling iteration, the degenerate in-
terval $[500, 500]$ is identified since a speed of $\infty$ is needed to finish $J_3$ in zero time.
Consequently, the new critical interval causes a monotonicity violation.

Building upon the above lemmas and observations, we propose an algorithm
called Time Overhead Earliest Deadline First (TOEDF), which eliminates both
monotonicity and execution violations as they occur, thus producing a feasible
schedule with a time overhead of arbitrary size. Figure 3.1 depicts the overall flow
of TOEDF, and Algorithm 2 gives the major steps.
Algorithm 2 $[T] = \text{TOEDF}(J, \Delta t)$

1: **Input:** The job set $J$, and the transition interval length, $\Delta t$.  
2: **Output:** A valid voltage Schedule $T = \{(T_i, s_i)\}$, consisting of a set of critical-interval/speed pairs.  
3: $i := 1; /*$ the critical interval index */  
4: **while** $J$ is not empty **do**  
5: Find the next critical interval $T_i = [t_s(i), t_f(i)]$ with $s_i = s(T_i)$;  
6: if $(i > 1 \text{ AND } s_i > s_{i-1})$ /* Monotonicity or execution violation */ **then**  
7: Restore the timing information from the previous iteration;  
8: Remove $(T_{i-1}, s_{i-1})$ from $T$;  
9: Merge the interval $T_i$ with $T_{i-1}$;  
10: $i -=; /*$ Roll back the critical interval index */  
11: **end if**  
12: Adjust the end points of interval $T_i$ to accommodate the transition interval $\Delta t$;  
13: Backup the timing information;  
14: Remove all jobs in $J_i$ from $J$;  
15: Insert $(T_i, s_i)$ into $T$;  
16: Squeeze $T_i$ into a single time point;  
17: $i++;$  
18: **end while**  
19: RETURN $T$;
Figure 3.1. The process used by TOEDF to find a valid voltage schedule.
In TOEDF, when a monotonicity or execution violation is encountered (line 6), 
\( T_{i-1} \) is extended to include all jobs scheduled in \( T_i \) (line 9). “Squeezing” a critical 
interval changes the timing parameters of some jobs (line 16). The parameters are 
required if \( T_{i+1} \) causes a violation, so we save these parameters (line 13). After \( T_i \) 
is identified, we may need to extend it like M-LPEDF to incorporate \( \Delta t \) (line 12). 
One must be particularly careful during this step. For example, if an end point of 
\( T_i \) is within \( \Delta t \) of any previous interval (say \( T_j \)) in \( T \), we should not extend \( T_i \) so 
that it overlaps \( T_j \). This will prevent TOEDF from including more time overhead 
than necessary. (This can be done by maintaining a proper data structure when 
implementing the algorithm. We omit the details.) In what follows, we apply 
Algorithm 2 to the job set in Figure 2.2(a) and discuss more details of TOEDF.

In the first iteration of Algorithm 2, the critical interval \([600, 1000]\) with \( s_1 = 1 \) is identified. The interval is extended to \([500, 1100]\) (with \( \Delta t = 100 \)) and then 
\((T_1, s_1)\) is inserted into \( T \). During the second iteration, \( J_3 \) causes an execution 
violation (Figure 3.2(a)). Therefore, \((T_1, s_1)\) is removed from \( T \) and \( J_3 \) is merged 
with \( J_2 \in J \). The new \( T_1 \) is expanded to \([500, 1050]\) to reflect the earliest release 
time and the latest deadline of both jobs \( J_2 \) and \( J_3 \), and is further expanded to \([400, 1150]\) to accommodate time overhead. \( T_1 \) keeps \( s_1 \) from the previous iteration and 
\((T_1, s_1)\) is once again inserted into \( T \).

In the third iteration, \( J_4 \) causes a monotonicity violation (Figure 3.2(b)). Therefore, \( T_1 = [400, 1150]\) is extended to \([350, 1300]\). Finally, during the fourth iteration 
another critical interval, \( T_2 = [0, 350]\) is identified with \( s_2 = s(T_2) = 100/350 = 0.29 \). 
Note that \( t_f(2) = t_s(1) = 350 \). \( T_1 \) already includes a transition interval from 350 
to 450, so \((T_2 = [0, 350], s_2)\) is inserted into \( T \) (we assume no transition at time 0.) 
The resulting schedule is depicted in Figure 3.2(c). One can readily verify that this 
schedule indeed guarantees the schedulability of all the jobs.
Figure 3.2. The job set from Figure 2.2(a), scheduled by TOEDF.
Theorem 1 Algorithm 2 always produces a valid voltage schedule in $O(n^3)$ time, given an initially schedulable job set.

Proof: Executing jobs $J_i \in T_i$ at $s(T_i)$ ensures that all deadlines within $T_i$ are met. Monotonicity and execution violations are eliminated by merging the current critical interval with the previous one and the resulting interval guarantees that all job deadlines are met, according to Lemmas 2 and 3. The dominating step per iteration is identifying the critical interval, which is $O(n^2)$. The outer loop can repeat up to $O(n)$ times, for a total time complexity of $O(n^3)$. □
CHAPTER 4

A UNIFIED ALGORITHM FOR NON-IDEAL DVS PROCESSORS

In the previous section, we present an algorithm called TOEDF that produces a feasible and reasonably energy efficient voltage schedule when time overhead is not negligible. In this section, we present an enhanced algorithm to improve the energy efficiency of TOEDF and incorporate discrete speeds and energy overhead simultaneously.

4.1 Improving the Basic Algorithm

Unnecessary energy may be wasted when using TOEDF since, in order to eliminate violations, we use “higher-than-necessary” speeds for some intervals. Although removing violations is critical for guaranteeing the feasibility of a voltage schedule, one should strive to shorten the intervals that demand higher speeds to reduce energy consumption. For example when applying TOEDF to the job set in Figure 2.2(a), the processor speed for the interval [400,1200] is 1.0. However, one can readily verify that using the processor speed of 1.0 during the interval [590,1200] can also guarantee the schedulability of the jobs scheduled in that interval, i.e., $J_2$, $J_3$, and $J_4$. Energy is saved in two ways: first, the interval length that demands the speed of 1 is reduced; second, extra time is available for any remaining jobs, such as $J_1$, which can be used to further reduce their required speed. In this case, $J_1$ can be executed at $100/490 = 0.2$, instead of 0.29 like the schedule in Figure 3.2 Therefore,
if we have to use a higher-than-necessary speed, we should restrict the usage of this speed to as short of an interval as possible. We formalize this idea in Problem 1.

**Problem 1** Given a set of jobs, $\mathcal{J}$, and a constant speed, $s^*$ (unless noted otherwise, $s^*$ is a constant speed equal to or higher than the minimum speed required to meet all deadlines of $\mathcal{J}$), find the shortest interval in which all deadlines of $\mathcal{J}$ are met.

The key to solving Problem 1 is to realize that we really have one degree of freedom, i.e., how long the start of interval can be delayed. Of course, by delaying the interval we run the risk of missing job deadlines. To prevent any deadline misses, we introduce the concept of the *latest start time* for a job set in Definition 1 and an important lemma stating how to compute it in Lemma 4.

**Definition 1** The Latest Start Time $t_{LS}$, for a job set $\mathcal{J}$, is the latest time at which jobs in $\mathcal{J}$ can begin execution at speed $s^*$ and still meet all deadlines.

**Lemma 4** The job set $\mathcal{J}$ scheduled by **EDF** has the latest start time,

$$t_{LS} = \min \{t_{LS}(i) | t_{LS}(i) = d_i - \sum_{J_k \in \text{hp}(J_i)} \frac{c_k}{s^*}, i = 1..|\mathcal{J}| \},$$

where $\text{hp}(J_i)$ is the job set containing jobs with priorities equal to or higher than the priority of $J_i$.

**Proof:** First we prove that starting at a time later than $t_{LS}(i)$ causes $J_i$ to miss its deadline. Then we prove that $J_i$ does not miss its deadline if execution begins at or before $t_{LS}(i)$.

1. By EDF, $\text{hp}(J_i)$ includes the jobs with deadlines no later than $d_i$. To guarantee that all jobs in $\text{hp}(J_i)$ meet their deadlines, there must be sufficient time to execute the cycles of these jobs. The total workload of jobs in $\text{hp}(J_i)$ is $W = \sum_{J_k \in \text{hp}(J_i)} c_k$, and the total time needed to execute this workload is $W/s^*$. It is trivial to see that starting later than $t_{LS}(i)$ will not give enough time to finish $W$. The minimum $t_{LS}(i)$ is the most restrictive latest start time of all jobs in $\mathcal{J}$, so it is the correct choice for $t_{LS}$.
(2) Suppose that beginning execution at \( t_{LS}(i) \) causes \( J_i \) to miss its deadline. Considering the execution of jobs in \( hp(J_i) \), there must be idle time in the interval \([t_{LS}(i), d_i]\) such that no job in \( hp(J_i) \) are executed during this idle time. Beginning execution at any time earlier than \( t_{LS}(i) \) will not alter the job released after the idle interval (including \( J_i \) based on EDF), so \( J_i \) must still miss its deadline. This contradicts the assumption in Problem 1 that all jobs in \( J \) executing at the speed \( s^* \) will finish by their deadlines. □

Lemma 4 provides a way to find \( t_{LS} \) for a given set of jobs and a constant speed \( s^* \). Then, with a simple plane-sweeping algorithm we find the earliest finish time \( t_{EF} \) of all jobs in \( J \) based on \( t_{LS} \). Together, \( t_{LS} \) and \( t_{EF} \) form a shortest time interval as defined in Problem 1. With these observations in mind, we construct an algorithm called MININT (Algorithm 3), which solves Problem 1.

\begin{algorithm}
\textbf{Algorithm 3} \([T] = \text{MININT}(J, s^*)\)
\begin{enumerate}
\item \textbf{Input:} \( J \), the job set sorted from highest to lowest priority and \( s^* \), the speed at which the jobs in \( J \) must execute.
\item \textbf{Output:} The minimum interval \( T = [t_s, t_f] \).
\item \( t_s := \infty \);
\item \( Sum := 0 \);
\item \textbf{for} \( (i := 1; i < |J|; i++) \) \textbf{do}
\item \hspace{1em} \( Sum := Sum + \frac{d_i}{s^*} \);
\item \hspace{1em} \( t_{LS}(i) := d_i - Sum \);
\item \hspace{1em} \( t_s := \min\{t_s, t_{LS}(i)\} \);
\item \textbf{end for}
\item \( t_f := \text{Identify\_Finish\_Given\_Start}(J, t_s, s^*) \);
\item \( T := [t_s, t_f] \);
\item RETURN \( T \);
\end{enumerate}
\end{algorithm}

To show that Algorithm 3 indeed produces a minimum-length valid interval, we present Theorem 2.

\textbf{Theorem 2} Given a set of jobs, \( J \), and a constant speed, \( s^* \), Algorithm \text{MININT} finds a minimum length interval \( T = [t_s, t_f] \) needed to complete every job \( J_k \in J \) at the speed \( s^* \) by its deadline.
Proof: We consider the following two cases based on the length of $T$:

(1) $|T| = \sum_{i=1}^{\lfloor \frac{c}{s} \rfloor}$

Based on the arguments used in proving Lemma 4, it follows that $|T|$ in this case is a shortest interval needed in order to complete all jobs in $J$.

(2) $|T| > \sum_{i=1}^{\lfloor \frac{c}{s} \rfloor}$

In this case there must be idle time within $T$. Without loss of generality, suppose there is one idle interval, $T_I = [t_c, t_r]$, where $t_c$ is the time when the previous job completes execution, and $t_r$ is the time when the next job is released. By Algorithm 3, $t_s = t_{LS}$, so $t_s$ cannot be increased. Moving $t_s$ back in time will only reduce $t_c$. Because $t_r$ is the release time of a job in $J$, it is fixed. The result is either $|T_I|$ increases or new idle intervals appear inside $T$, causing $|T|$ to increase. For the fixed start time $t_s$ and the constant speed $s^*$, the finish time of the final job, $t_f$, is also fixed. Expanding $T$ beyond $t_f$ will introduce idle time and moving it back will cause the final job to miss its deadline. Therefore, moving $t_s$ or $t_f$ to any other point cannot decrease $|T|$ without causing a deadline miss.

Note that for case (2), the minimum interval found by Algorithm 3 is the only possible minimum interval, while for case (1) the placement of the minimum interval is not unique. This completes the proof. □

Algorithm MININT can be readily incorporated into TOEDF to improve its energy performance. In TOEDF, when a violation occurs and jobs are merged with those executed in the previous critical interval, we can apply MININT to find the shortest interval within which every job deadline is satisfied. This typically results in increased energy savings. Moreover, we show in the following sections that MININT
can also help to improve the energy efficiency when dealing with discrete voltage levels and energy overhead.

4.2 Discrete Voltage/Frequency Levels

Until now we have assumed that the processor speed can be continuously varied. However, current commercial DVS processors [3, 11, 15] only have a finite number of speeds. This factor must be integrated into voltage scheduling algorithms to provide a valid and energy efficient schedule.

One intuitive way to deal with discrete speeds is to round up the required frequency and voltage to some allowed level. Unfortunately, this can be extremely pessimistic and energy inefficient, especially for many commercial processors with only a few voltage levels available [3]. A better approach is proposed in [16] that can use the two levels immediately above and below the desired voltage/speed value to optimally schedule a single job. Kwon et. al. [19] built upon the results in [16] and develop an optimal voltage/frequency scheduling algorithm for an entire job set, called AllocVT. Although theoretically optimal, AllocVT is not practically applicable on real processors because excessive voltage transitions, roughly two per job, are introduced. This, coupled with the omission of time overhead, will cause jobs to miss their deadlines.

To consider both time overhead and discrete speeds simultaneously, we believe that it is more advantageous to incorporate the discrete speed effects into the construction of critical intervals and let it propagate to future critical interval construction. Specifically, after a critical interval is identified in Algorithm 2, its speed is increased to the next available level. Recall that when a higher-than-necessary speed is applied, we can use MININT to find the minimal interval needed for the given job set and speed. However, when increasing the speed to the next higher level, it can
introduce a significant amount of unused idle time even after we apply algorithm MININT to find a minimal length interval. A better method to utilize these idle times to save energy is to relax the requirement that all jobs originally found in the critical interval must run at the higher speed (note that Lemma 3 only holds when the speed increase is due to a monotonicity or execution violation). Therefore, we keep only one of these busy intervals (intervals without idle time) for the final voltage schedule, with the expectation that the rest of the jobs may benefit from the higher-than-necessary speed assignment for this interval and can execute at a lower speed. We summarize this approach in Algorithm 4.

**Algorithm 4** \([T'_i, s'_i, J'_i] = \text{DISCRETE} (T_i, s_i, J_i, S)\)

1: **Input:** The initial critical interval \(T_i\), its speed \(s_i\), the set of jobs \(J_i\) in \(T_i\), and the set of valid speeds \(S\).
2: **Output:** A valid critical interval \(T'_i\), running at \(s'_i\) that executes jobs \(J'_i\).
3: \(s'_i := \min\{S_j \in S | S_j \geq s_i\}\);
4: Identify one busy interval, \(B(i) = [t_s, t'_s] \in T_i\);
5: \(T'_i := B(i)\);
6: \(J'_i := \text{all jobs } J_k \in J_i \text{ that finish execution in } B(i)\);
7: **RETURN** \(T'_i, s'_i\) and \(J'_i\);

In Algorithm 4, one problem is how to select which busy interval to keep (line 4). A good choice can lead to low computation cost and higher energy efficiency. There are a number of heuristics, such as always selecting (1) the first, (2) the last, (3) the shortest, or (4) the longest busy interval. Though each of these approaches has its intuitive advantages, none of them dominates the others in our experiments. This is due to the many patterns of job arrival times, deadlines, and execution cycles. Therefore, we simply choose the first busy interval as it is the most computationally convenient.
4.3 Transition Energy Overhead

So far, we have ignored energy overhead in our voltage scheduling algorithms. Similar to time overhead, we account for energy overhead while constructing the critical intervals. This allows the effect to propagate throughout the schedule. Our approach works as follows: when a new critical interval, $T_i$, is identified, whether or not this critical interval is kept depends on whether or not the energy consumed by adopting its speed is smaller than that consumed by merging it with an adjacent critical interval, $T_j$. (The term adjacent is defined in section 2.1). The idea is that if the energy transition overhead is so significant that using different processor speeds for different intervals consumes more energy than using a single speed, we can simply merge adjacent critical intervals and remove the associated voltage transitions.

To merge $T_i$ with $T_j$ while guaranteeing the schedulability of the jobs in both intervals, we adopt $s_j$ for the new interval’s speed. $T_j$ was identified before $T_i$ and has a higher speed according to Lemma 1. Now we have the same situation as Problem 1, so MININT is used to find a minimum-length interval.

4.4 The Unified Algorithm

By combining the techniques from sections 4.1 through 4.3 with Algorithm 2, a valid voltage schedule with superior energy savings is produced while accounting for practical limitations of real-world DVS processors, including time and energy overhead and discrete voltage levels. We call this unified algorithm UAEDF (Algorithm 5). The overall flow of UAEDF is given in Figure 4.1.

Algorithm 5 follows the same general flow as Algorithm 2. First, it identifies the next critical interval assuming any speed is valid (line 6) and matches the next valid speed using DISCRETE (line 7). Then it removes monotonicity or execution violations and shortens the critical interval with algorithm MININT (lines 9-12).
Figure 4.1. Flow diagram of UAEDF.
Algorithm 5 \([T] = \text{UAEDF}(\mathcal{J}, \Delta t, S)\)

1: **Input:** \(\mathcal{J}\), the job set, \(\Delta t\), the transition interval length, and \(S\) the set of valid speeds.

2: **Output:** A valid voltage Schedule \(\mathcal{T} = \{(T_i, s_i)\}\), consisting of a set of critical-interval/speed pairs.

3: \(i := 1; /* \text{The critical interval index.} */\)

4: \(i_{\text{prev}} := 1; /* \text{The index of the previously inserted critical interval} */\)

5: **while** \(\mathcal{J}\) is not empty **do**

6: Find the next critical interval \(T_i = [t_s(i), t_f(i)]\) with \(s_i = s(T_i)\);

7: \([T_i, \mathcal{J}_i, s_i] := \text{DISCRETE}(T_i, \mathcal{J}_i, s_i, S)\)

8: if \((i > 1 \text{ AND } s_i > s_{i_{\text{prev}}})\) then

9: Remove \(T_{i_{\text{prev}}}\) from \(\mathcal{T}\);

10: \(\mathcal{J}_{i_{\text{prev}}} := \mathcal{J}_{i_{\text{prev}}} \cup \mathcal{J}_i\)

11: \(T_{i_{\text{prev}}} := \text{MININT}(\mathcal{J}_{i_{\text{prev}}}, s_{i_{\text{prev}}})\);

12: \(i -= -;\)

else

13: \(i_{\text{prev}} := i;\)

end if

16: Adjust the end points of interval \(T_{i_{\text{prev}}}\) to accommodate \(\Delta t\);

17: \(\mathcal{T}_{\text{adj}} := \text{the set of all intervals adjacent to } T_{i_{\text{prev}}};\)

18: Find \(T_k \in \mathcal{T}_{\text{adj}}\) such that merging \(T_{i_{\text{prev}}}\) with \(T_k\) has the maximal gain in energy savings;

19: if the maximal gain > 0 then

20: \(\mathcal{J}_{i_{\text{prev}}} := \mathcal{J}_{i_{\text{prev}}} \cup \mathcal{J}_k;\)

21: \(T_{i_{\text{prev}}} := \text{MININT}(\mathcal{J}_{i_{\text{prev}}}, s_k);\)

22: Adjust the end points of interval \(T_{i_{\text{prev}}}\) to accommodate \(\Delta t\);

23: Replace \((T_k, s_k)\) in \(\mathcal{T}\) with \((T_{i_{\text{prev}}}, s_{i_{\text{prev}}});\)

24: \(i_{\text{prev}} := k;\)

25: \(i -= -;\)

else

27: Insert \((T_{i_{\text{prev}}}, s_{i_{\text{prev}}})\) into \(\mathcal{T};\)

28: end if

29: Squeeze \(T_{i_{\text{prev}}}\) into a single time point;

30: \(i++;\)

end while

32: **RETURN** \(\mathcal{T};\)
A greedy approach is adopted in Algorithm 5 to deal with the energy overhead during the voltage transition (lines 18-25). After a critical interval is found, we check to ensure whether or not it is more advantageous to merge it with an adjacent interval (lines 18-19). If such a merge is necessary (line 19), we simply merge it with the adjacent interval that will lead to the maximal energy savings, and then apply MININT to minimize the interval length (line 20-25). Finally, as in TOEDF, the newly generated interval is ”squeezed” into one single time point (line 29) and one iteration of the algorithm is completed. Theorem 3 formally states that Algorithm 5 finds a valid schedule in $O(n^3)$ time.

**Theorem 3** Algorithm 5 always produces a valid voltage schedule with a time complexity of $O(n^3)$.

**Proof:** Theorem 1 states that TOEDF always produces a valid voltage schedule. Algorithm 5 includes time overhead and handles the same violations as TOEDF, but uses MININT instead of simply matching release times and deadlines. It follows from Theorems 1 and 2 that this method always produces a valid schedule. Energy overhead is also accounted for using MININT, so Theorem 2 ensures the validity of this step as well. Discrete voltage levels are matched by rounding up to a valid level and taking the first busy interval. The jobs in the busy interval will obviously meet their deadlines, and the rest of the jobs will be rescheduled by methods we have just shown to be valid.

The most time consuming step per iteration is identifying the next critical interval in $O(n^2)$ time, which can repeat $O(n)$ times. Therefore, the overall time complexity is $O(n^3)$. □
CHAPTER 5

EXPERIMENTAL RESULTS

In this section, we quantify the impact of transition overhead and discrete voltage levels and evaluate the energy savings of our proposed algorithms with both the randomly generated job sets and real-world examples. We compare our algorithms against Sys-Clock, presented in [32], modified to schedule EDF job sets rather than RM task sets. As far as we know, Sys-Clock is the only previous work that can include an arbitrarily large time overhead and still guarantee a valid voltage schedule.

5.1 Setup

In our experiments, we base our power model on the AMD Mobile Athlon4 DVS processor [3]. The Mobile Athlon4 can run at voltage levels in the range 1.2 to 1.4 V with 50 mV steps and corresponding frequencies of 500 MHz to 1 GHz with 100 MHz steps. The valid voltage levels for various Athlon4 processors are given in Table 5.1. For experiments with more than 5 voltage levels, we interpolate these performance points using a second order polynomial (see Figure 5.1). The minimum speed/voltage for all experiments is 500 MHz at 1.2 V. Power is modeled using equation 1.2. The value of $C_{SW}$ was set to 12.79 nF as shown in Table 5.3. We derived this value using Table 5.2 from AMD’s datasheet [3]. A sample derivation for the 1 GHz Athlon4 is given in equations 5.1 through 5.3. All other values in Table 5.3 were also derived using equation 5.2.
\[ P = V_{DD} \times I = C_{SW}V_{DD}^2f_{clk} \]  \hspace{1cm} (5.1)

\[ C_{SW} = \frac{V_{DD} \times I}{V_{DD}^2f_{clk}} \]  \hspace{1cm} (5.2)

\[ C_{SW} = \frac{(1.40)(17.90)}{(1.40)^2(1 \times 10^9)} = 12.79 \times 10^{-9} \]  \hspace{1cm} (5.3)

The CPU power ranges from 9.2 to 25 W. The system includes a low power “sleep” state that the processor can utilize when idle. We assume that it takes \( \Delta t/2 \) time to enter the sleep state, and another \( \Delta t/2 \) to exit. The power consumed while in the sleep state is 2.4 W.
TABLE 5.1. PERFORMANCE SETTINGS OF AMD’S MOBILE ATHLON 4

<table>
<thead>
<tr>
<th>$V_{DD}$ (V):</th>
<th>1.400 (MHz)</th>
<th>1.350 (MHz)</th>
<th>1.300 (MHz)</th>
<th>1.250 (MHz)</th>
<th>1.200 (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Speed (MHz)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>850</td>
<td>850</td>
<td>700</td>
<td>600</td>
<td>500</td>
<td>≤ 500</td>
</tr>
<tr>
<td>900</td>
<td>900</td>
<td>700</td>
<td>600</td>
<td>500</td>
<td>≤ 500</td>
</tr>
<tr>
<td>950</td>
<td>950</td>
<td>800</td>
<td>700</td>
<td>600</td>
<td>≤ 500</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>800</td>
<td>700</td>
<td>600</td>
<td>≤ 500</td>
</tr>
<tr>
<td>1100</td>
<td>1100</td>
<td>1000</td>
<td>800</td>
<td>700</td>
<td>≤ 600</td>
</tr>
<tr>
<td>1200</td>
<td>N/A</td>
<td>1200</td>
<td>1000</td>
<td>900</td>
<td>≤ 600</td>
</tr>
</tbody>
</table>
TABLE 5.2. CORE VOLTAGE/CURRENT ON AMD’S MOBILE ATHLON 4

<table>
<thead>
<tr>
<th>Frequency/State</th>
<th>Voltage (V)</th>
<th>Maximum $I_{cc}$ (A)</th>
<th>Die Temperature (degrees Celsius)</th>
</tr>
</thead>
<tbody>
<tr>
<td>850 MHz</td>
<td>1.40</td>
<td>15.71</td>
<td>95</td>
</tr>
<tr>
<td>900 MHz</td>
<td>1.40</td>
<td>17.14</td>
<td>95</td>
</tr>
<tr>
<td>950 MHz</td>
<td>1.40</td>
<td>17.14</td>
<td>95</td>
</tr>
<tr>
<td>1000 MHz</td>
<td>1.40</td>
<td>17.90</td>
<td>95</td>
</tr>
<tr>
<td>1100 MHz</td>
<td>1.40</td>
<td>17.90</td>
<td>95</td>
</tr>
<tr>
<td>1200 MHz</td>
<td>1.35</td>
<td>18.50</td>
<td>95</td>
</tr>
<tr>
<td>Halt/Stop Grant C2</td>
<td>1.20</td>
<td>2.00</td>
<td>95</td>
</tr>
<tr>
<td>Stop Grant C2</td>
<td>1.20</td>
<td>1.07</td>
<td>50</td>
</tr>
<tr>
<td>Stop Grant C3/S1</td>
<td>1.20</td>
<td>0.80</td>
<td>50</td>
</tr>
</tbody>
</table>
### TABLE 5.3. POWER CHARACTERISTICS OF AMD’S MOBILE ATHLON 4

<table>
<thead>
<tr>
<th>Max Frequency (MHz)</th>
<th>Max Voltage (V)</th>
<th>Maximum $I_{CC}$ (A)</th>
<th>MaxPower (W)</th>
<th>$C_{sw}$ (nF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>850</td>
<td>1.40</td>
<td>15.71</td>
<td>21.99</td>
<td>13.20</td>
</tr>
<tr>
<td>900</td>
<td>1.40</td>
<td>17.14</td>
<td>24.00</td>
<td>13.60</td>
</tr>
<tr>
<td>950</td>
<td>1.40</td>
<td>17.14</td>
<td>24.00</td>
<td>12.89</td>
</tr>
<tr>
<td>1000</td>
<td>1.40</td>
<td>17.90</td>
<td>25.06</td>
<td>12.79</td>
</tr>
<tr>
<td>1100</td>
<td>1.40</td>
<td>17.90</td>
<td>25.06</td>
<td>11.62</td>
</tr>
<tr>
<td>Transition</td>
<td>1.35</td>
<td>18.50</td>
<td>2.4</td>
<td>16.67</td>
</tr>
</tbody>
</table>
Time overhead is modeled by a constant value in the range of 0 (no time overhead) to 5 ms. Note that while most DVS processors have a time overhead in the range of tens to hundreds of microseconds, these numbers may be misleading as they ignore synchronization delay with off chip components such as memory, which can be quite large. The Compaq iPAQ, for example, requires 20 ms to synchronize with its main memory after a frequency switch [10]. Unless both the program and data memory requirements of a given job are small enough to fit in the processor’s cache, this synchronization time overhead cannot be ignored. Energy overhead is modeled using $\Delta E = \Delta E_{DC} + \Delta E_{CPU}$ where $\Delta E_{DC}$ is the energy consumed by the DC-DC converter, and $\Delta E_{CPU}$ is the energy consumed by the CPU during a transition. $\Delta E_{DC} = \eta C_{DD}(V_{DD1} - V_{DD2})$ with $C_{DD} = 100\mu F$ and $\eta = 0.9$ as presented by Burd in [5]. $\Delta E_{CPU} = k \Delta t$ where $k = 2.4$ W is the power consumed in the stop-grant...
state of the AMD processor (entered during a transition) according to [3].

5.2 Results

We first constructed and tested 100 randomly generated sets of 20 jobs. The jobs are assumed to always require the worst-case execution cycles to complete and release times are uniformly distributed between [0,800] and [0,1000] µs, respectively. The relative deadlines of the jobs are normally distributed with an average of 810 µs and a standard deviation of 280 µs. We scheduled each job set using Sys-Clock, TOEDF, and UAEDF on a processor with 1, 14, 5, and 2 discrete voltage levels. The energy from each schedule was normalized against the optimal LPEDF schedule without transition overhead on a continuous processor, i.e., we used LPEDF as the lower bound on energy. The results are summarized in Figure 5.2. Note that the energy of each algorithm saturates, so we do not extend the graphs past 2.5 ms.

In each figure, we can see that energy increases rapidly with increased time overhead and fewer discrete voltage levels. Clearly both our algorithms are better in most cases and at least as good as Sys-Clock in other cases. At low time overhead, between 0 and 200 µs, TOEDF and UAEDF are within 2% of each other on a CPU with 5 or more levels. At 2 levels UAEDF outperforms TOEDF by as much as 10%. Past 200 µs, UAEDF outperforms both algorithms by as much as 25%. Both TOEDF and Sys-Clock saturate past 1.5 ms, while UAEDF maintains a 6 to 10% improvement.

We also apply our algorithms to three real-world examples: (1) A Computerized Numeric Controller (CNC) task set based on the work by Kim et al. in CNC [17], (2) An Avionics task set based on Locke’s work in [24] and (3) a video phone task set based on the work by Shin et al. in [33]. Each task set was first converted to a job set by unrolling the tasks to their respective hyper-periods. These results are
given in Figures 5.3, 5.4 and 5.5. In contrast with the random job sets, the energy of real-world examples does not increase monotonically with $\Delta t$. This is because Figure 5.2 represents the average of 100 job sets, while the real-world examples represent one specific job set each. For a particular job set, some values of $\Delta t$ may be “nicer” than others, as the impact of $\Delta t$ depends on the timing parameters of the given jobs.

For the CNC example in Figure 5.3, both TOEDF and UAEDF save energy when compared to Sys-Clock at low $\Delta t$; from 7.5% on a continuous processor to 25% with two levels and $\Delta t < 200\mu s$. For each set of levels, the three algorithms saturate at $\Delta t > 1.4$ ms and the $\Delta t$ at which TOEDF and UAEDF separate becomes smaller. For a continuous processor in Figure 5.3(a), TOEDF and UAEDF are within 2% until 1.7 ms. The $\Delta t$ value at which TOEDF and UAEDF become significantly different decreases to 1.6 ms for 14 levels, and 0.5 ms for 5 and 2 voltage levels. Also, the $\Delta t$ at which UAEDF saturates to Sys-Clock becomes larger with fewer discrete voltage levels; from 2.7 ms on a continuous processor to 3.7 ms with 2 discrete levels. The maximum energy savings of UAEDF compared to TOEDF change from 7% on a continuous processor to about 35% with only two voltage levels. For the CNC job set, we can conclude that the fewer the number of voltage levels, the more benefit we get from UAEDF. This is because there are several large intervals which require less than the minimum speed of 0.5, so UAEDF can schedule a larger number of cycles at the smaller speed than TOEDF for larger $\Delta t$.

For the avionics job set in Figure 5.4, the behavior is the opposite of that observed in the random and CNC job sets in that the energy savings of UAEDF does not increase with every reduction in the number of available voltage levels. This is because almost all cycles must execute at speeds greater than $S_{\min}$. With high $\Delta t$ and only two levels, all cycles execute at $S_{\max}$, and the savings from UAEDF
come from clustering idle times with MININT and entering the sleep state. This 
amounts to less than 2% savings. The maximum energy savings of 7.9% occurs with 
14 levels and a $\Delta t$ between 0.5 and 1.4 ms. TOEDF is comparable to UAEDF until 
$\Delta t = 0.9$ ms with continuous or 14 voltage levels and until $\Delta t = 0.2$ with 5 voltage 
levels.

The results on the video phone job set are displayed in Figure 5.5. For the 
continuous, 14 and 5 voltage level processors, the energy increase above LPEDF is 
minimal, no more than 3.6%. For these processors the energy consumption of all 
three algorithms is comparable. This is because time overhead has less of an impact. 
Jobs in this job set have very large active intervals ($\gg 5$ ms). Thus even a large 
time overhead does not cause many monotonicity or execution violations. As the 
number of levels decreases, the sensitivity of energy consumption to time overhead 
also increases (due to rounding up speeds). Hence the large increase in energy with 
only two levels. An interesting observation is that the processor with 5 discrete 
levels consumes less energy the one with 14 levels. This is because each interval 
identified by LPEDF for the video-phone job set has a normalized ideal speed near 
0.74. The next level higher on the 14 level processor is 0.79, while on the 5 level 
processor it is 0.75. For the 2 voltage level processor, UAEDF outperforms LPEDF 
in the range of 1 ms to 3.5 ms by up to 5%, and outperforms Sys-Clock by as much 
as 7%. After 3.5 ms, UAEDF and TOEDF converge, each saving close to 5% of the 
energy consumed above LPEDF when compared to Sys-Clock.

An important point not illustrated in our figures is that the difference between 
including and excluding energy overhead in our experiments was minimal (less than 
1% change in energy consumption). There are three main reasons for this result. 
First, our algorithms already handle time overhead by reducing the number of tran-
sitions which naturally reduces the impact of energy overhead also. Second, the
energy overhead that is not directly dependent on $\Delta t$, i.e. that incurred by the DC-DC converter (maximally $(0.9)(100 \times 10^{-6})|1.4^2 - 1.2^2| = 0.0468 \text{ mJ}$) is less than 1% of the energy consumed by a 1 ms interval executing at even the minimum power level (9.2 mJ). Typically critical intervals are longer than 1 ms. Third, assuming that avoiding a transition to eliminate an instance of energy overhead does save energy, the savings are balanced by the execution of some cycles at a higher speed instead of a lower one. This is particularly true for fewer discrete voltage levels.
Figure 5.3. Increase in energy above LPEDF for the CNC job set.
Figure 5.4. Increase in energy above LPEDF for the Avionics job set.
Figure 5.5. Increase in energy above LPEDF for the Video Phone job set.
CHAPTER 6

SUMMARY

6.1 Conclusions

In this paper, we studied the impact that practical limitations of current processors can have on the energy consumed in a hard real-time system. These limitations include time and energy overhead and discrete voltage levels. We have shown through examples and analysis that transition time overhead can cause a theoretically optimal schedule to become invalid if not correctly accounted for during the scheduling process. Accounting for time overhead is not a trivial matter, as trying to make adjustments to the optimal schedule by inserting overhead between voltage intervals will likely cause jobs to miss their deadlines.

We presented two algorithms, TOEDF and UAEDF, which always produce valid voltage schedules given an initially schedulable job set. These are the first such algorithms for the off-line preemptive EDF scheduling problem studied in this paper. UAEDF and TOEDF are comparable for small $\Delta t$ in the range of 100 $\mu$s to 0.5 ms. As $\Delta t$ becomes large, UAEDF can reduce energy consumption by as much as 38% in our experiments. For very large $\Delta t$, i.e., greater than 3 to 5 ms, both UAEDF and TOEDF saturate at the minimum constant voltage level that completes all jobs by their deadlines. This is the method used by Sys-Clock on fixed priority systems. For a very small time overhead, i.e., $\Delta t < 100 \mu$s, both TOEDF and UAEDF are within 1% of the theoretical lower bound with at least 5 voltage levels.
Generally, our results show that all three limitations are closely related. The sensitivity of the energy consumption to time overhead increases with fewer discrete voltage levels. Energy overhead is generally less significant with increased time overhead, because to form a valid schedule with a larger time overhead, one must reduce the number of transitions. Also, energy overhead is already insignificant on the AMD Athlon4 model used for our experiments. We believe this is because the Athlon processor was originally designed for performance rather than energy efficiency, and then the DVS features were added to create the Mobile Athlon.

6.2 Future Work

Currently, the optimality of our algorithms is not guaranteed, so further algorithm development may improve results even more. For example, UAEDF always pushes the minimum intervals as far back as possible (section 4.1). A more intelligent placement of the minimum interval could reduce the overall energy consumption. If the minimum interval has only one valid placement, it is very likely that it contains idle time that can be utilized by unscheduled jobs that overlap that interval. How to account choose which overlapping jobs to include in the idle time must also be investigated. When matching a discrete voltage level, UAEDF currently keeps only the first busy interval, which in some cases isn’t the best choice. How to optimally select which interval to keep is another open question.

For the scheduling process to give a practical voltage schedule for an even wider range of systems, we will need to account for other implementation details, such as context switching overhead, support for different transition models and other priority schemes, such as fixed priority scheduling. Future work must address these limitations.
APPENDIX A

TASK SETS USED IN COMPUTER SIMULATIONS

In this section we describe how each job set was generated and used in our experiments. The random job set in Table A.1 was generated for the initial test of our algorithms. The selection of [0, 1000] μs for the release times was meant to keep the jobs clustered tightly together, so we could observe interesting interactions between them. The worst case execution cycles were uniformly distributed in [0, 800] μs so that the intensity of any individual critical interval would not exceed 1 (the normalized maximum), but could fall below 0.5 (the normalized minimum speed). The relative deadlines were chosen so that the active interval of every job is on average around 8 times larger than 100 μs, the time overhead of the Athlon4. We chose 20 jobs per set, because this is slightly more than the number of tasks that could be simultaneously active in our practical examples. As mentioned earlier, we generated 100 job sets with these parameters, and the experiments average the results from each.

The remaining three task sets were converted to job sets by expanding each task out to the LCM of every period in the system. The CNC task set in Table A.2 was expanded without modification. The video-phone task set in Table A.3 did not initially specify task deadlines, which were assumed to be equal to the task period. We moved each deadline back to the value shown. For a task set where every deadline is equal to the period and a preemptive EDF policy is used, there is only one critical
interval when using LPEDF to schedule the tasks, which is a relatively uninteresting schedule because there are no voltage/frequency transitions. In general, the period of each task can be any value, so we chose deadlines between 0.5 and 1 times the period for each task.

We added deadlines to the Avionics task set in Figure A.4 just like the videophone set. In addition, a phase shift was added to some of the jobs, which means that the release/deadline of each job instance of that task was moved forward in time by the phase shift amount. In addition, task 8 was omitted from our experiments. This is because the period is 59 µs greatly increases the LCM of the system, thus increasing the number of jobs and the scheduling time to about 1 hour. If we were actually interested in using our schedules in a real system, we would only need to run our algorithms one time and we would keep task 8. For our purposes, however, we are only interested in estimating the energy requirements. Because a large number of data points were required for our estimation, task 8 was omitted.
### TABLE A.1. STATISTICS FOR THE RANDOMLY GENERATED JOB SETS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>\mathcal{J}</td>
</tr>
<tr>
<td>$r_i$</td>
<td>Uniformly distributed in [0, 1000] μs</td>
</tr>
<tr>
<td>$c_i$</td>
<td>Uniformly distributed in [0, 800] μs</td>
</tr>
<tr>
<td>$d_i$</td>
<td>$r_i +$ normally distributed value: $avg = 810 \mu s$, $std = 280 \mu s$</td>
</tr>
</tbody>
</table>

### TABLE A.2. THE COMPUTERIZED NUMERIC CONTROLLER TASK SET

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET (ms)</th>
<th>Period (ms)</th>
<th>Deadline (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.035</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td>2</td>
<td>0.040</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td>3</td>
<td>0.180</td>
<td>4.8</td>
<td>4.8</td>
</tr>
<tr>
<td>4</td>
<td>0.720</td>
<td>4.8</td>
<td>4.8</td>
</tr>
<tr>
<td>5</td>
<td>0.165</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td>6</td>
<td>0.165</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td>7</td>
<td>0.570</td>
<td>9.6</td>
<td>4.0</td>
</tr>
<tr>
<td>8</td>
<td>0.570</td>
<td>7.8</td>
<td>4.0</td>
</tr>
</tbody>
</table>

### TABLE A.3. DEADLINE MODIFIED AUDIO AND VIDEO PHONE TASK SET

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET (ms)</th>
<th>Period (ms)</th>
<th>Deadline (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50.386</td>
<td>66.667</td>
<td>66</td>
</tr>
<tr>
<td>2</td>
<td>9.826</td>
<td>66.667</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>1.844</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>1.386</td>
<td>40</td>
<td>35</td>
</tr>
</tbody>
</table>
### TABLE A.4. DEADLINE/PHASE SHIFT MODIFIED AVIONICS TASK SET

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET (ms)</th>
<th>Period (ms)</th>
<th>Deadline (ms)</th>
<th>Phase Shift (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.051</td>
<td>1</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3.000</td>
<td>200</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2.000</td>
<td>25</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>5.000</td>
<td>25</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1.000</td>
<td>40</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>3.000</td>
<td>50</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>5.000</td>
<td>50</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8.000</td>
<td>59</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>9</td>
<td>9.000</td>
<td>80</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>2.000</td>
<td>80</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>5.000</td>
<td>100</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>3.000</td>
<td>200</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1.000</td>
<td>200</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1.000</td>
<td>200</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>3.000</td>
<td>200</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1.000</td>
<td>1000</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>1.000</td>
<td>1000</td>
<td>1000</td>
<td>0</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


