

# Energy Efficient Voltage Scheduling for Real-Time Operating Systems

Trevor Pering

EECS Department  
University of California, Berkeley  
Berkeley, CA 94704  
pering@eecs.berkeley.edu

Prof. Robert Brodersen

EECS Department  
University of California, Berkeley  
Berkeley, CA 94704  
rb@eecs.berkeley.edu

## Abstract

*This paper applies the concept of real-time process scheduling to a Dynamic Voltage Scaling (DVS) microprocessor. DVS allows a microprocessor to save energy by operating at the optimal voltage for the task at hand. Efficient operation requires a new class of algorithms which we term voltage schedulers. The necessary foundation for these algorithms is presented along with the foreseen implementation difficulties.*

## 1. Introduction

The microprocessor is a significant power drain in many portable systems. For CMOS design, the energy-per-operation of a microprocessor is given by the equation

$$E_{op} \propto CV^2$$

where  $C$  is the switching capacitance and  $V$  is the operating voltage. There are three ways to reduce the energy consumed by a microprocessor: reduce the number of operations performed, reduce the switched capacitance of each operation, and reduce the operating voltage. This paper discusses the framework for reducing energy by run-time optimization of the operating voltage.

The maximum clock speed of a microprocessor is given by the relation

$$f_{max} \propto \frac{V-c}{V}$$

where  $V$  is the operating voltage from the previous equation and  $c$  is a constant. Combining these two equations, assuming a constant capacitance, yields

$$f_{max} \propto \frac{\sqrt{E_{op}} - c}{\sqrt{E_{op}}}$$

This equation states that there is a trade-off between energy and delay: we can decrease the energy consumed by slowing the processor clock and reducing the operating voltage [1].

A simple example of the energy/delay trade-off starts with a system that operates at 100 Mhz, 3.3V, and 50% processor utilization. Reducing the operating voltage to 2.4V, necessitating a clock reduction to 50 Mhz, would decrease the energy consumption by 47%. This reduction in clock fre-

quency does not impact functionality because the system was originally 50% idle. Our model assumes the processor consumes no energy when idle.

It is important to understand that merely changing a processor's clock frequency is not an effective technique for reducing energy consumption. Reducing the clock frequency will reduce the power consumed by a processor; however, it does not reduce the energy required to perform a given task. Lowering the voltage along with the clock actually alters the energy-per-operation of the microprocessor, reducing the energy required to perform a fixed amount of work.

*Dynamic Voltage Scaling* (DVS) is a technique that allows a *voltage scheduler* routine to alter a microprocessor's operating voltage at run-time. The voltage scheduler analyses the state of the system and determines the optimal target voltage. One possible technique for the voltage scheduler is to use task deadlines, a concept common in real-time operating systems, to determine the extent to which a task can be lengthened.

This paper applies the fundamentals of real-time deadline scheduling to the voltage scheduling problem. Tasks considered are aperiodic and specified by  $\{S_i, C_i, D_i\}$  where  $S_i$  indicates the task start time,  $C_i$  the computational resources required, and  $D_i$  the task deadline. Start times and deadlines are specified as absolute times; computational resources required are specified as execution time with the processor running at full speed.

## 2. Voltage Scheduling Graphs

This section describes *Voltage Scheduling Graphs* which are used to depict the interaction between voltage schedules and tasks.

On a voltage scheduling graph, shown in Figure 1, the X-axis represents time and the Y-axis represents processor speed. With these axes, the computation required by a task is proportional to the 'area under the curve' for that task. Processor speed is normalized to the range [0...1].

The height of a task (processor speed) and work performed ( $C_i$ ) determine its energy consumed,  $E_i$ . Figure 2 contains two tasks with identical computation requirements: their areas are equivalent. Task  $T_1$  executes at a higher processor speed and voltage than Task  $T_2$ , as a result it consumes more energy.

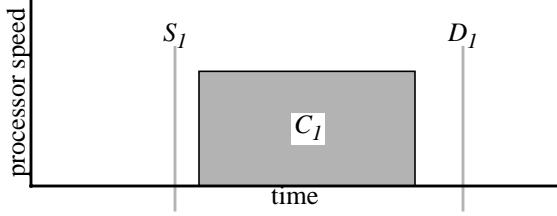


Figure 1: The Voltage Scheduling Graph

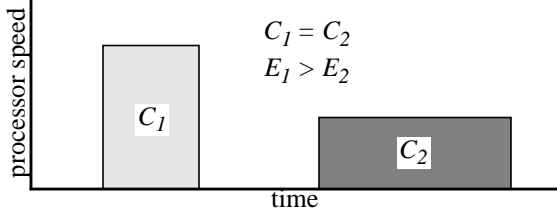


Figure 2: Task energy vs. computation

For simplicity, we discuss voltage schedules in terms of processor speed and not voltage settings. It is important to remember, however, that a change in processor speed implies a corresponding change in operating voltage.

Because relationships with respect to voltage are nonlinear,  $E_i$  is not simply  $C_i$  multiplied by the processor speed. For our purposes, fortunately, it is sufficient to realize that a flat schedule, i.e. one where all tasks are scheduled at the same speed and voltage, will be more efficient than one with varying speed settings. The concept is similar to a sum-of-squares optimization with  $X_i$  representing the scheduled processor speed: given the constraint  $\sum X_i = c$ , the quantity  $\sum X_i^2$  is minimized when  $X_1 = X_2 = \dots = X_n$ .

### 3. Voltage Scheduling Basics

We define an optimal voltage schedule to be one for which all tasks complete on or before deadline and the total energy consumed is minimized. Multiple identical tasks, where  $S_i = 0$  and  $D_i = a \sum C_k$  with  $a \geq 1$  for all tasks, are optimally scheduled by a constant processor speed  $P = 1/a$ . This is an extreme case; real systems will rarely, if ever, have tasks with identical start times and deadlines.

Figure 3 depicts three tasks with  $S_i = 0$  and differing deadlines. For such a schedule, with ordered deadlines  $D_i \leq D_k \forall i < k$ , the optimal voltage schedule can be found with the following algorithm, which runs in  $O(n)$  time to schedule  $n$  tasks. The intermediate workload,  $W_i$ , determines the optimal schedule for all tasks with deadlines at or before  $D_i$ .  $P_i$  calculates the processor speed to use such that future deadlines are not violated.

- Let  $W_i = \frac{1}{D_i} \sum_{k=1}^i C_k$
- Let  $P_i = \text{MAX}(W_k, k \geq i)$
- For any time  $T$ , find the minimum  $j$  s.t.  $T \leq D_j$ ; schedule the processor at speed  $P_j$ .

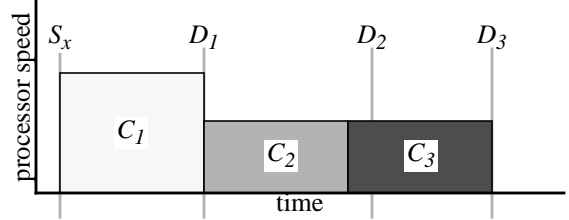


Figure 3: Optimal coincident task scheduling

Extending this algorithm to the general case where  $S_i \neq S_k \forall i, k$  is non-trivial. A simple modification of the above algorithm, which considers a task only if  $S_i \leq T$  and re-evaluates at any time  $T = S_k$ , has the drawback of initially under-estimating the resources required, as in Figure 4. This greedy algorithm also has the potential to schedule tasks such that they no longer meet their deadlines by initially running too slowly.

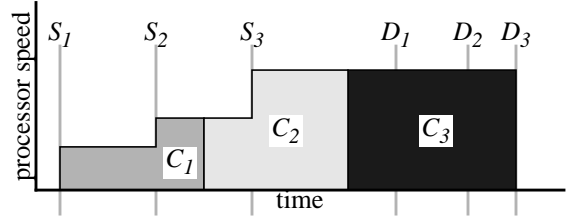


Figure 4: Greedy algorithm under-estimation

Voltage scheduling specifies the speed at which a processor should run for a given interval, not necessarily which task should run when. Vertical layering, shown in Figure 5, is used when several tasks are potentially scheduleable during the same time interval. One of the standard thread scheduling techniques, such as earliest-deadline-first or time-sliced, can then be used to schedule the tasks within that interval. The vertical ordering of tasks is not significant.

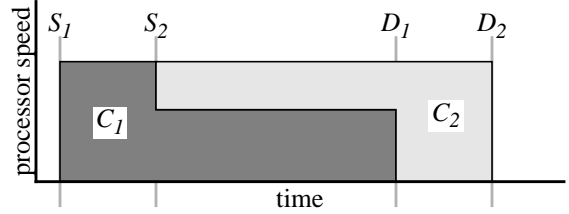


Figure 5: Vertical Layering

### 4. Optimal Scheduling

We have developed a proof of optimality for voltage schedules. A complete description of the proof is beyond the scope of this paper; the important result, however, is summarized by the following lemma:

**For a region spanned by a given task specification, each point in time will either be scheduled at the minimum speed spanned by that task or else the task will not be scheduled to run at that point.**

Figure 4 is therefore not an optimal schedule because tasks  $T_1$  and  $T_2$  are not always scheduled at the minimum speed spanned by their start and deadline times.

Algorithms are currently under development to optimally schedule a given set of tasks. Our current working algorithm incrementally adds a new task to an existing schedule in  $O(n^2)$  time, resulting in an overall complexity of  $O(n^3)$  to schedule a  $n$  tasks. This algorithm has not yet been fully implemented and it is unclear if it will be a tractable solution. Appendix A gives an example of this algorithm being applied to a simple voltage schedule.

## 5. Task Specification Variance

The work presented in the previous section assumes a complete and accurate task specification. In a real system, however, these two assumptions may not be valid. First, prior knowledge of a task's start time may be unavailable. Second, the computational resources requested by a task will only be an estimate of the actual resources required.

Heuristic estimates can be applied to reserve computation for unspecified future tasks being introduced into a schedule. For the greedy algorithm, the intermediate workload,  $W_i$ , could be increased by an estimate based on the expected processing needs of blocked tasks. Our optimal algorithm can be augmented by inserting predicted placeholder tasks. Both these techniques will unavoidably produce sub-optimal schedules due to the variance of task specification estimation.

The computational resources requested by a task,  $C_i$ , are only an estimate of the actual resources required. Variance can be introduced, for example, by cache behavior, code behavior, and interaction with other threads. In a system without DVS, such variance is not always a crucial consideration: functionality is not affected as long as the processor is fast enough. Typical real-time systems specify  $C_i$  as a *worst-case* computation time. A DVS system using worst-case specifications, however, will typically schedule tasks at higher than necessary speeds, increasing their energy consumption.

Specifying  $C_i$  as the average computation time for soft real-time systems is desirable to minimize energy. Sometimes, then, a task will miss its deadline. In such situations it is necessary to determine the fate of the task: does it continue execution, and if so, at what speed? A method for communicating the desired behavior is needed as the 'correct' behavior is application dependent.

For hard real-time systems missed deadlines are unacceptable. In this case, the worst-case computation time can be used, resulting in a schedule that meets timing constraints but is not always energy-optimal.

## 6. Related Work

[5] first presented the idea of voltage scaling within the context of general-purpose microprocessor systems. Their

algorithms use interval based scheduling: each fix-sized time interval runs at a constant processor speed. The processor speed used is based on the activity of previous intervals. This technique has the advantage of an easy implementation, but the disadvantage of sub-optimal results.

In [4], we apply cycle-level simulations of the algorithms of [5] to several deadline-based applications, such as MPEG and audio stream processing. Our results indicate that interval-based voltage scheduling is effective, realizing up to a 70% reduction in system energy. However, some applications fall significantly short of optimal. Additionally, the efficiency realized is extremely dependent on the interval length, making it difficult to choose one interval such that all applications are scheduled effectively.

[3] presents the design of a processor core which uses voltage scaling to run at the minimum voltage necessary for operation at a given input clock frequency. Their design demonstrates the feasibility of voltage scaling, but does not allow direct software control over the processor speed.

Our research group is currently under development of a DVS microprocessor with silicon expected September 1998. We estimate our processor will consume 1.8mW at 8MHz/1.1V and 220mW at 100MHz/3.3V in a 0.6 $\mu$ m process. We implement the ARM8 instruction set with a 16kB unified on-chip cache.

In our system, a voltage scheduler can set the target clock frequency through the co-processor at a resolution of 1 MHz. The operating voltage is then set by a feedback loop which compares the current and target frequencies. Expected transition time is  $\sim 10\mu$ s, worst-case.

## 7. Summary

This paper presents a foundation for energy-efficient Dynamic Voltage Scaling (DVS) microprocessors using concepts found in real-time operating systems. Several algorithms, both sub-optimal and optimal, are discussed for the voltage scheduling problem. We plan to implement these algorithms on a DVS microprocessor that is currently under design.

Implementation details, such as the variance in task computation times, are discussed. Voltage scheduling introduces additional variance into the completion time of a task which impacts the performance of soft and hard real-time systems.

## Acknowledgments

*This work was funded by DARPA and made possible by cooperation with Advanced RISC Machines Ltd (ARM). The authors would like to thank Eric Anderson for his help on, and proof of, the optimal scheduling algorithm.*

## References

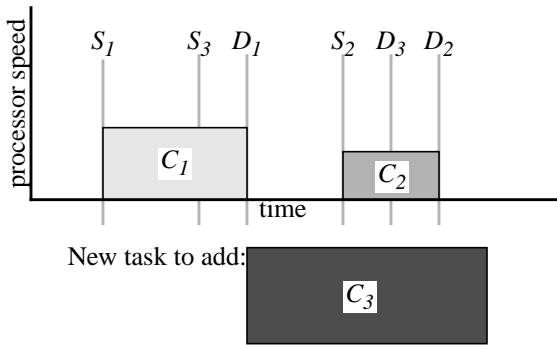
- [1] T. Burd and R. Brodersen, "Energy Efficient CMOS Microprocessor Design," *Proc. 28th Hawaii Int'l Conf. on System Sciences*, Vol. 1, Jan. 1995.

- [2] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*, second edition, Addison-Wesley, 1997.
- [3] T. Kuroda, et. al., "Variable Supply-Voltage Scheme for Low-Power High-Speed CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, Vol. 33, No. 3, March 1998.
- [4] T. Pering, T. Burd, and R. W. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," *Int'l Symp. on Low Power Electronics and Design*, August 1998.
- [5] M. Weiser, "Some computer science issues in ubiquitous computing," *Communications of the ACM*, Vol. 36, pp. 74-83, July 1993.

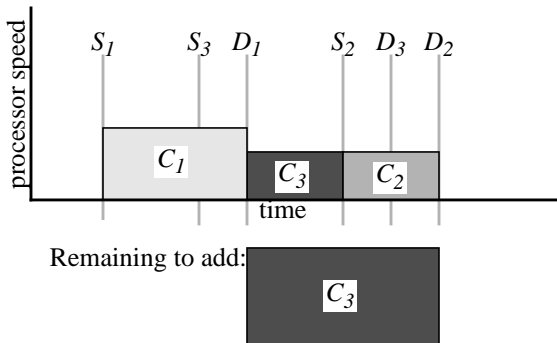
**Appendix A: Voltage Scheduling Example**

This section gives an example of our current algorithm for creating an optimal voltage schedule. It is based on the discussion in Section 4. The task block placed below the time axis represents the new task, or portion thereof, that has yet to be scheduled. Complexity analysis:

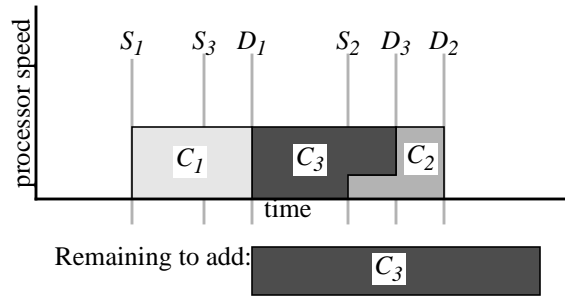
- $n$  tasks to schedule
- $O(n)$  speed settings to consider for each task
- $O(n)$  linked tasks requiring adjustment for each setting
- Total complexity:  $O(n^3)$



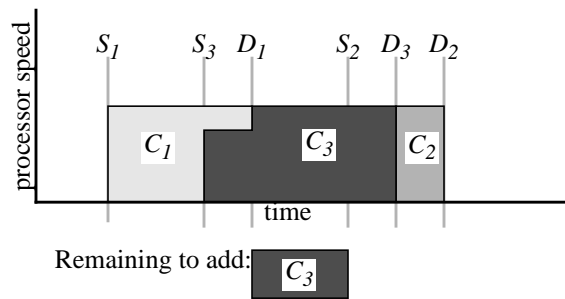
**Step 0: Initial schedule and new task ( $T_3$ ) to add.**



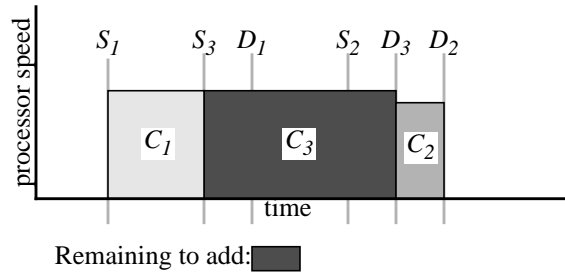
**Step 1: Fill in idle time spanned by  $T_3$ . Link  $T_2$  to  $T_3$  because  $T_2$  intersects  $T_3$  and they are scheduled at the same speed setting.**



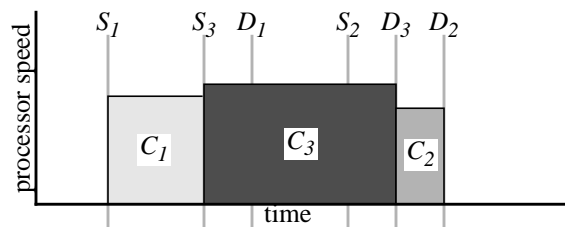
**Step 2: Increase the speed of linked tasks ( $T_2, T_3$ ) to the minimum speed of adjacent task  $T_1$ . Link  $T_1$  to  $T_3$  because it is spanned by  $T_3$  and scheduled at an equal speed.**



**Step 3: Increase the speed of linked tasks ( $T_1, T_2, T_3$ ) until  $T_2$  is completely pushed-out of the range spanned by  $T_3$ .  $T_2$  is no longer considered 'linked' to  $T_1$ .**



**Step 4: Increase the speed of linked tasks ( $T_1, T_3$ ) until  $T_1$  is completely pushed-out of the range spanned by  $T_3$ .  $T_3$  is then no longer linked to any other task.**



**Step 5: Add remaining computation to schedule.**