Dynamic Task-Level Voltage Scheduling Optimizations*

Jeffrey A. Barnett

Northrop Grumman Corporation

Abstract

Energy versus delay tradeoffs are explored for systems that must manage energy expenditure as well as computation deadlines. The focus is execution of a single process on a single processor. Two probabilistic process models are considered along with a family of power dissipation models. The first process model assumes that process complexity is exactly c cycles with probability $p(c)$. The second model considers the detailed branching and loop structure of the code. Probabilities are attached at branch points. The power models assume that energy dissipation per cycle is proportional to v^m and that execution time for a cycle is proportional to v^{-n}, where v is supply voltage. The energy versus delay tradeoff is implemented using dynamic voltage and clock adjustments. The problems solved include 1) minimize expected execution time given a hard energy budget and 2) minimize expected energy expenditure given a hard deadline. The problem of minimimizing the expected value of $Q(E, T)$, where Q is a penalty function and E and T are, respectively, total energy and total time, is also solved using the first process model. Analysis determines theoretical conditions where it may be profitable to switch voltage or modify an a priori voltage schedule.

Keywords: Energy-aware systems, energy management, time management, dynamic voltage scheduling, agile voltage scheduling, power management points.

*Full version appears in IEEE Transactions on Computers 54(5), 2005, pp. 508–520.

1 Background, Summary, and Contributions

1.1 Background

Resource management has always been a primary focus area within computer science. Virtually all pragmatic models of computation contain a layer—the operating system—dedicated to it. The resources managed typically include the CPU, memory, secondary storage, and device access. Object controllers and other systems that interact with their environments have special resource-management constraints that are called hard (or soft) realtime requirements. Properties of, and algorithms for realtime systems and resource management are among the most studied in computer science.

Energy is a relatively recent addition to the list of resources that must be managed. This is driven, in part, by increased computation capabilities and miniaturization that entail heat density and dissipation problems [38]. Thus, energy management problems exist for large systems such as servers [30, 36, 37, 38, 39] as well as portable devices [12, 29, 30, 37]. Another driver is economics: energy costs money.

There are other classes of systems that must be radical more power aware. Two examples are space systems [42] and unattended, distributed sensor networks [47]. The Sky Tower aircraft with an endurance goal of six months using electric engines [2] is a third example. These systems use batteries as buffers but must scavenge energy from the environment to achieve long endurance. Solar panels are one potential energy source; vibration, chemical, and ambient heat are other possibilities. Energy availability is not constant so optional activities must be scheduled for times of plenty, e.g., when solar panels are illuminated, and all but the most urgent needs must be deferred at other times. Many such systems use single thread architectures to conserve and control resource expenditures.

Several approaches to energy management are currently being pursued. Many researchers are investigating hardware improvements that will reduce energy consumption with minimal or no impact on runtime performance. Examples include use of two different supply voltages within the same chip [20], use of two identical processors where each has a different supply voltage [45], dynamical control of L2 cache line size [25], distribution within a super scalar chip design [49], use of FPGA devices [33], and design of bus structures that use less energy [28]. These approaches promise "automatic" savings because the hardware is more energy efficient. Several studies [25, 31, 49] propose metrics to measure the ultimate success

of these investigations. Proposed minimization objectives have the form ET^{α}, where E is the energy used by a test computation, T is its execution time, and α is a constant.

Another approach—the one investigated herein—provides controls used by the application and/or the operating system to effect an energy versus performance tradeoff at runtime. This possibility is interesting because many, if not most, energy-constrained systems also have temporal constraints [3, 32]. The available controls are the supply voltage and the clock frequency. When the voltage is increased, the clock frequency can be increased. However, the energy cost per unit of computation will also increase. Several commercially available processors provide voltage/clock controls: Intel Xscale and Strongarm [21], Transmeta Crusoe [44], AMD K6-2+[1], and the IBM 405PL [23] and are some examples. The Advanced Configuration and Power Management (ACPI) specification provides standard interfaces for low-power states and energy and thermal controls as well as plug-and-play hardware protocols [22].

Generally, researchers following the program pursued herein concentrate on minimimizing energy consumption given hard or soft temporal constraints [3, 12, 16, 17, 29, 30, 32, 36, 37, 39, 38, 41, 45, 48]. Of these, only Cao [12] and Qiu and Pedram [38] note that the dual problem—minimize expected execution time given an energy budget—is also significant. Both formulations are addressed below.

1.2 Summary

Section 2 introduces the power models and the process models used for algorithm development and analysis. The power models specify relations between supply voltage levels and processor speed and rate of energy consumption. The process models encode probabilistic knowledge about process complexity measures in cycles. Two process models are examined: The *simple probabilistic model* provides the function p, where $p(x)$ is the probability that process complexity is exactly x cycles. A *structural process model* provides a flow graph of the process. The associated metrics are the complexities of the simple segments (the graph nodes) and the probabilities of branching from one segment to another one.

Section 3 formulates a set of optimization problems and solves them analytically using a specific simple power model, then Section 4 provides the optimal solutions for the same optimizations using arbitrary power models. Two optimizations—find the voltage schedule that minimizes expected execution time given a not-to-exceed energy budget and find the voltage

schedule that minimizes expected energy consumption given a not-to-exceed deadline—are solved for both the simple probabilistic and structural power models. A third optimization—find the voltage schedule that minimizes the expected value of $Q(E, T)$, where E is total energy expenditure, T is total computation time, and Q is a general penalty function—is solved for the simple probabilistic process model.

Section 5 discusses methods for gathering process metrics—the probabilities and complexity estimates that comprise process models—and how to insert that information in application systems. The probabilities are available from general application knowledge, domain models, and feedback from system executions. The complexity measures, on the other hand, are generated from machine models, cycle-level simulators, and compiler analyses.

Section 6 analyzes the optimality results. Optimal strategies are compared to strategies that predict average-case behavior. The optimal strategies always behave as though future complexity will be worse than average. Other analyses discuss the value of using updated complexity information promptly and the robustness of the optimal algorithms to parameter estimation errors. Finally, it is shown that voltage levels in optimal strategies can only change at points where something new is learned about future complexity of the process. At all other points, voltage remains constant. Section 7 provides the conclusions.

1.3 Contributions

The focus of this article is how best to use voltage and clock controls to effect application-driven energy versus delay tradeoffs. The investigation assumes a single process or a dependent set of tasks executing on a single processor. This restriction is imposed for two reasons: The first is that many energy-constrained systems are organized this way [41]. The second reason is that exact, basic optimization methods are needed as the building blocks for schedulers that handle multiple processes executing on multiple processors. Gruian [17] notes that one of the main reasons behind using task-level scheduling techniques is that they can lower the energy consumption of a certain task without requiring strong knowledge about the other tasks in the system and one can also use classic scheduling techniques at system level and still get energy efficiency.

Several results developed below appear to be novel. The first is the solution of optimization problems where energy is the constrained resource and expected execution time is to be minimized. The second is the formulation and solution to minimizing the expected value

of the general penalty function $Q(E, T)$, where E is energy consumption and T is execution time. The third contribution is the development of optimal techniques to schedule processes from their structural models. Perhaps the most significant result is an analysis that determines when an optimal voltage schedule will change voltage and hence computation speed and energy consumption rate, and when these parameters will remain constant: voltage level will only change in an optimal schedule when something *new* is learned about the future complexity of the process.

2 Power and Process Models

Section 2.1 introduces the power models and Section 2.2 introduces the process models used for the optimizations performed in Sections 3 and 4.

2.1 Power Models

Many modern computers permit voltage and clock frequency to be modified by the executing program. Some examples are the Intel Xscale and Strongarm [21], the Transmeta Crusoe [44], the AMD K6-2+ [1], and the IBM 450PL [23]. Increased voltage permits increased clock frequencies to be used. The penalty for faster computations is increased energy consumption per unit of computation. That unit is the cycle. Cycles are used to measure execution progress rather than instructions because cycles are reasonably similar to one another in terms of energy dissipation and execution time while instructions are not [17, 29, 32, 41, 48].

The general models for the energy expenditure and time to execute c cycles, respectively, are $e = \alpha c v_{dd}^m$ and $t = \beta c v_{dd}/(v_{dd} - v_T)^{n+1}$, where v_{dd} is supply voltage (the variable) and v_T is threshold voltage [10, 14, 40]. The constants v_T, α, β, m, and n depend on the architecture. Martin [31] discusses "smooth circuits" where v_T scales with v_{dd} so that $t = \beta' c/v_{dd}^n$ would represent execution time, and many authors [16, 29, 37, 41, 45], while acknowledging the effects of v_T, ignore it either in optimizations, examples, or testing. Manzak and Chakrabarti [30] note that when $v_{dd} > 3v_T$, only a 0.1% error is introduced if $v_T = 0$ is assumed, i.e., the voltage schedule developed ignoring v_T uses only 0.1% more energy than an optimal schedule. For these reasons, the power model can be simplified to $e = \alpha c v^m$ and $t = \beta c/v^n$, where $v = v_{dd}$. A further notational simplification follows if the *units* of e and t are chosen to entail $e = cv^m$ and $t = c/v^n$.

Two limitations on speed adjustments should be noted: 1) voltage and clock levels are discrete and 2) minimum and maximum levels are imposed by the hardware. For example, the Transmeta Crusoe [44] provides as many as two clock frequencies for a single voltage level and 36 combinations. However, Gutnik and Chandrakasan [18] and Namgoang et al [35] discuss architectures where continuous speed adjustments are possible and Melhem et al [32] note that systems which are able to operate on a (more or less) continuous voltage spectrum are rapidly becoming a reality thanks to advances in power supply engineering and CPU design. Chandrakasan et al [13] shows that a few voltage/speed levels are sufficient to achieve almost the same energy savings as infinite levels.

The development below follows Lorch and Smith [29] and Shin et al [41] by assuming that voltage can vary continuously. However, many studies [16, 32, 48] do deal with issues of discrete and limited settings and their work should be consulted when these effects are important in particular applications. Techniques to deal with limitations on minimum and maximum voltage settings are discussed elsewhere [3, 48].

It should also be noted that changing processor speed through voltage and/or clock adjustments may exact time and energy penalties [11, 21, 36, 39, 44, 48]. Lorch and Smith [29] assume that changing voltage incurs little or no overhead and Aydin et al [3] suggest, as will be assumed here, that these penalties simply be added to the complexity estimates of process segments and otherwise be ignored during optimization. Some computer components may also depend on constant power availability. It appears that power-aware architectures tend to arrange independent supplies in these cases so that there are better controls over resource consumption. Thus, these concerns are peripheral to the optimizations developed herein.

The power models appearing in the optimizations developed below are denoted by Π_{mn}. Using the Π_{mn} model, the energy to execute c cycles at constant voltage v is $e = cv^m$ and the execution time is $t = c/v^n$. Some optimizations will assume that voltage is *agile*, i.e., voltage can vary continuously. In such cases,

$$e = \int_0^c v(x)^m \, dx \qquad t = \int_0^c v(x)^{-n} dx, \tag{1}$$

where $v(x)$ is the voltage used when cycle x is executed. The Π_{21} model is the one most often used [3, 16, 30, 37, 45, 48]. Lorch and Smith [29] also do basic analysis with Π_{21} but note that formulas developed using Π_{11} better fit simulation results. Shin et al [41] also develop optimizations using Π_{21} but evaluate an example architecture with a $\Pi_{2(.7)}$

model. All optimizations blow are initially performed with the Π_{11} model in order to simplify derivations. Section 4 re-presents the results using general Π_{mn} models.

Melhem et al [32] proves, for a variety of models, that the lowest energy utilization to execute a fixed number of cycles when there is a hard deadline is achieved by the constant voltage solution that uses all of the available time. Ishihara and Yasuura [24] note the same result and Błazewicz et al [7] note that when the rate of consumption of some resource is a convex function of CPU speed, an ideal schedule will run each task at a constant speed. A corollary is used below: the least time to execute a fixed number of cycles when there is a hard energy constraint is achieved by the constant voltage solution that uses all of the available energy.

2.2 Process Models

The unit presented for scheduling is the process. Two types of process models are defined below. The first is the simple probabilistic model where all that is known about a process is that its complexity is x with probability $p(x)$. The second is the structural model where the branching behavior and probabilities of the branches are specified. A simple probabilistic model can be derived from structural information though there is an information loss—so better optimizations are generally possible with the latter.

2.2.1 The Simple Probabilistic Process Model

The simple probabilistic process model is specified by a nonnegative function, p, where $p(x)$ is the probability that execution will terminate after exactly x cycles. Two assumptions are made: 1) $p(x) = 0$ if $x \leq 0$ and 2) there exists a finite c such that $p(x) = 0$ if $x > c$. In the notation used below, c will always denote the *smallest* value for which $\int_0^c p(x)\,dx = 1$.

The function $P(x) = \int_0^{x-} p(y)\,dy$ is the probability that process complexity is less than x and $z(x) = \int_x^c p(y)\,dy$ is the probability that the complexity is at least x. Thus, $z(x) = 1 - P(x)$. A simple probabilistic process model will often be specified by P rather than p. Though the information content is equivalent, the former is often more straightforward to elicit and represent.

Both [16] and [29] introduce simple probabilistic models and find voltage schedules that minimize expected energy consumption given a not-to-exceed deadline. The latter uses their

results to improve performance of an existing dispatch schedule where the deadlines are already defined. The value of using statistical information in the form of the simple model is demonstrated by analysis and simulation [17].

2.2.2 The Structural Process Model

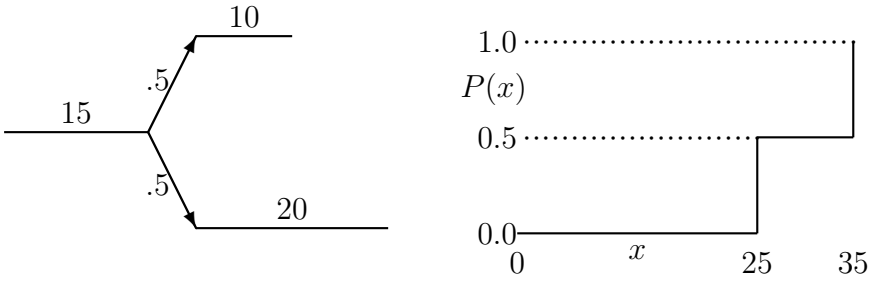
The structural model of a process provides details of its branching. A process is represented by a 4-tuple, $(\sigma, S, c_{\mathrm{x}}, p)$, where S is a set of code segments, $c_{\mathrm{x}}(s)$ is the complexity measured in cycles of each $s \in S$, and σ is the initial segment (entry point) of the process. Branch probabilities are represented by p, where $p(s_1, s_2)$ is the probability that execution of s_2 will immediately follow execution of s_1. Let $p(s) = \sum_{r \in S} p(s, r)$, then clearly $p(s) \leq 1$ is required for all $s \in S$. If $p(s) < 1$, then $1 - p(s)$ is the probability that an execution of s will be process terminal. Define $\theta(s)$ to be the set of all $r \in S$ such that $p(s, r) > 0$.

A directed graph can be used to represent a structural process model. The $s \in S$ are the nodes of the graph and there is a directed edge from each $s \in S$ to each member of $\theta(s)$. The nodes are labelled by $c_{\mathrm{x}}(s)$ and the arc from s to $r \in \theta(s)$ is labelled with $p(s, r)$. For the nonce, assume that process graphs are acyclic. Methods that deal with graph cycles will be discussed in Section 3.3 where loops are analyzed. Note, there can be multiple paths to as well as from a segment.

Many investigators use flow-graph technology to represent processes that will be voltage scheduled to minimize energy consumption given computational deadlines. The simplest case is a graph where the edges represent simple order dependencies [45]. Both [32] and [41] use graph models where probabilities annotated branches and worst- and average-case complexities are calculated for the continuations starting at each node. Zhu et al [48] present a generalization of an AND/OR model [15] where AND nodes represent parallel dispatch, OR nodes, where synchronization is forced, represent conditional branching, and probabilities annotate these branches. Worst- and average-case complexity statistics are used to improve voltage schedules within an innovative slack-stealing scheme. The formalism introduced here, like those in [32, 41, 48], has no direct method to represent common subroutines. However, a routine graph may be macro expanded, perhaps multiple times, into the parent graph.

2.2.3 Process Model Comparison

Consider a process with three segments. The first segment, with $c_x = 15$, is executed then there is a branch to one of two final segments: one with $c_x = 10$ and the other with $c_x = 20$. Figure 1(a) shows the structural model of this process with the assumption that both branch



(a) Structural model. (b) Simple process model.

Figure 1: The structural and simple models for the same process.

probabilities are 0.5. Figure 1(b) shows $P(x)$ for the simple probabilistic model of the same process. In the simple model $p(25) = .5$, $p(35) = .5$, and all other p-values are zero. The relevant observation, in this case, is that the structural model is more informed: total process complexity is known in the structural model at cycle 15 when the branch is selected. The simple model only provides new complexity information at cycle 25 where the process either halts or continues for 10 more cycles. As will be shown below, the delay in knowledge capture increases the expected energy and time measures that we are trying to minimize.

3 Optimizations

Optimized voltage schedules are derived using simple probabilistic and structural process models assuming the Π_{11} power model. The optimization results are presented for the general Π_{mn} model in Section 4. Table 1 summarizes the optimizations performed in this section, where $\mathrm{Ex}(\cdot)$ is the expected value operator, E is energy expenditure, and T is computation time. Other sorts of optimization criteria have been of recent interest. One example is reward-based scheduling [4] where core parts of processes must be scheduled. Reward values are assigned to the unscheduled process parts and the goal of the scheduler is to maximize its reward by scheduling these additional parts without exceeding resource limitations.

Sections 3.1.3 and 3.1.5 provide optimization examples, Section 3.2.3 describes algorithms

Table 1: Section 3 optimization summary.

Section	Model	Budget	Minimize
3.1.1	Simple	E	$\mathrm{Ex}(T)$
3.1.2	Simple	T	$\mathrm{Ex}(E)$
3.1.4	Simple	—	$\mathrm{Ex}(Q(E,T))$
3.2.1	Structural	E	$\mathrm{Ex}(T)$
3.2.2	Structural	T	$\mathrm{Ex}(E)$

to implement optimizations with structural models, and Section 3.3 discusses the modelling and handling of loops, including cases where the number of iterations is not bounded at compile time.

3.1 Simple Probabilistic Model Optimizations

3.1.1 Hard Energy Bound

The problem is to find the agile voltage schedule that minimizes the expected execution time of a process given its simple probabilistic model, p, and a not-to-exceed energy budget E. The expected execution time is $\int_0^c z(x)/v(x)\,dx$ because $z(x)$ is the probability that the process will still be executing at cycle x and $v(x)^{-1}$ measures the time to execute that cycle. The energy constrain is $E = \int_0^c v(x)\,dx$. This problem is solved using a Lagrangian multiplier and variational methods [9]. The objective has the form,

$$\int_0^c \frac{z(x)}{v(x)}\,dx + \lambda\left[E - \int_0^c v(x)\,dx\right],$$

where λ is the Lagrangian multiplier. Substitute $v(x) + \beta g(x)$ for $v(x)$, differentiate with respect to β, let $\beta = 0$, and set the result to zero:

$$-\int_0^c \frac{g(x)z(x)}{v(x)^2}\,dx - \lambda\int_0^c g(x)\,dx = 0$$
$$\int_0^c g(x)\left[\frac{z(x)}{v(x)^2} + \lambda\right]dx = 0.$$

Since $g(x)$ can be an arbitrary function, $\lambda + z(x)/v(x)^2 = 0$ is necessary. Therefore, $v(x) = kz(x)^{1/2}$ for some constant k. From $E = \int_0^c v(x)\,dx$, it follows that $k = E/I$, where $I = \int_0^c z(x)^{1/2}\,dx$. Thus, a summary of the minimization result is

$$v(x) = \frac{E}{I}\,z(x)^{1/2} \qquad \mathrm{Ex}(T) = \frac{I^2}{E} \qquad \mathrm{Ex}(E) = \frac{E}{I}\int_0^c z(x)^{3/2}\,dx, \tag{2}$$

where $\mathrm{Ex}(T) = \int_0^c z(x)/v(x)\,dx$ is the expected process execution time as noted above and $\mathrm{Ex}(E) = \int_0^c z(x)v(x)\,dx$ is expected energy utilization. Note that $\mathrm{Ex}(E) \leq E$ because $0 \leq z(x) \leq 1$.

3.1.2 Hard Time Bound

The problem is to find the agile voltage schedule that minimizes the expected energy utilization of a process given its simple probabilistic model, p, and a not-to-exceed time budget T. The method to solve this problem is identical to the one used in Section 3.1.1. The summary of this minimization is

$$v(x) = \frac{I}{Tz(x)^{1/2}} \qquad \mathrm{Ex}(T) = \frac{T}{I}\int_0^c z(x)^{3/2}\,dx \qquad \mathrm{Ex}(E) = \frac{I^2}{T}. \tag{3}$$

This result, for a different power model, also appears in [16] and [29].

3.1.3 Examples for Hard Energy and Time Bounds

Actual deployment of the results developed above may require numerical methods especially when p is a measured rather than an analytic function. The p used in the example below was chosen so that closed-form results could easily be generated.

Let $p(x) = \frac{\pi}{2c}\sin(\frac{\pi}{c}x)$ when $0 \leq x \leq c$ and $p(x) = 0$ elsewhere be a simple probabilistic model. In the first problem, E is a hard energy bound and in the second, T is a hard time bound. Table 2 presents the optimal agile voltage solutions and expected resource

Table 2: Optimal solution for hard energy and time bounds.

Budget	$v(x)$	$\mathrm{Ex}(E)$	$\mathrm{Ex}(T)$
E	$\dfrac{\pi E\cos\left(\dfrac{\pi}{2c}x\right)}{2c}$	$\dfrac{2}{3}E$	$\dfrac{4c^2}{\pi^2 E}$
T	$\dfrac{2c}{\pi T\cos\left(\dfrac{\pi}{2c}x\right)}$	$\dfrac{4c^2}{\pi^2 T}$	$\dfrac{2}{3}T$

consumptions for both problems. It is interesting to note that voltage is a decreasing function when energy is constrained and an increasing function when time is constrained. This will always be the case as an examination of (2) and (3) will reveal. The rising effect was

previously noted [29] and led to an approach named Processor Acceleration to Conserve Energy (PACE).

3.1.4 General Penalty Function

The problem is to find the agile voltage schedule that minimizes the expected value of $Q(E,T)$, where Q is a penalty function and E and T are, respectively, total energy consumption and total execution time, given a simple probabilistic model, p. The objective to minimize is, thus,

$$\mathrm{Ex}(Q(E,T)) = \int_0^c p(x)Q(E(x),T(x))\,dx,$$

where $E(x) = \int_0^x v(y)\,dy$ and $T(x) = \int_0^x v(y)^{-1}\,dy$. This problem is tackled by variational methods [9] starting with the substitution $v(x) + \beta g(x)$ for $v(x)$, differentiating with respect to β, letting $\beta = 0$, and equating the result to zero:

$$\int_0^c p(x)\left[Q_1\int_0^x g(y)\,dy - Q_2\int_0^x \frac{g(y)}{v(y)^2}\,dx\right]\,dx = 0,$$

where $Q_1 = \partial Q/\partial E$ and $Q_2 = \partial Q/\partial T$. Change the order of integration to produce the equivalent

$$\int_0^c g(x)\,dx\int_x^c p(y)\left[Q_1 - \frac{Q_2}{v(x)^2}\right]\,dy = 0.$$

Therefore, minimizing/stationary voltage schedules must satisfy

$$v(x)^2 = \frac{\int_x^c p(y)Q_2\,dy}{\int_x^c p(y)Q_1\,dy} \tag{4}$$

since g can be arbitrarily chosen.

Integral equations such as (4) are notoriously difficult to solve. The illustrative example in the following section was generated to demonstrate analytic rather than numerical solution techniques. However, some cases are straightforward. For example, there is often a constant v that satisfies (4) if $Q_2/Q_1 = f(E/T)$ for a suitably well-behaved f and Π_{11} is assumed. Let v be a constant, then

$$\begin{aligned} f(E/T) &= f\left(\frac{\int_0^x v\,dy}{\int_0^x v^{-1}\,dy}\right) \\ &= f(v^2), \end{aligned}$$

which is a constant. Therefore, $Q_2/Q_1 = f(v^2)$ and, from (4),

$$\begin{aligned} v^2 &= \frac{\int_x^c p(y) f(v^2) Q_1 \, dy}{\int_x^c p(y) Q_1 \, dy} \\ &= f(v^2). \end{aligned}$$

So the solutions of $v^2 = f(v^2)$, i.e., the square root of the fixed points of f, if any, are stationary solutions of (4). These conditions are met whenever $Q(E,T) = g(ET)$ or $Q(E,T) = g(\alpha E^r + \beta T^r)$ as simple computations will show.

3.1.5 General Penalty Function Example

Let $p(x) = c^{-1}$ when $0 \le x \le c$ and let $p(x) = 0$ elsewhere. Then p is a simple probabilistic process model. Let the penalty function Q have the form $Q(E,T) = E + \frac{3}{2}c^2(1 - e^{-2T})$ so that $Q_1 = 1$ and $Q_2 = 3c^2 e^{-2T}$. Therefore, a stationary solution must satisfy

$$\begin{aligned} v(x)^2 &= \frac{\int_x^c p(y) Q_2 \, dy}{\int_x^c p(y) Q_1 \, dy} \\ &= \frac{\int_x^c 3ce^{-2T} \, dy}{\int_x^c \frac{1}{c} \, dy} \\ &= \frac{3c^2 \int_x^c e^{-2T} \, dy}{c - x}. \end{aligned}$$

The above is satisfied by $v(x) = c - x$, where $T(y) = \log \frac{c}{c-y}$, as can be verified by substitution.

3.2 Structural Process Model Optimizations

3.2.1 Hard Energy Bound

The problem is to find a voltage schedule that minimizes expected execution time given the structural process model, $(\sigma, S, c_{\mathrm{x}}, p)$, and a not-to-exceed energy budget E. Define the quantity

$$I(s) = c_{\mathrm{x}}(s) + \sqrt{\sum_{r \in \theta(S)} p(s,r) I(r)^2} \tag{5}$$

for each $s \in S$. Note that $I(s) = c_{\mathrm{x}}(s)$ when s is a terminal segment because $\theta(s) = \emptyset$ and, hence, the summation is vacuous.

It is shown here that the optimal voltage, $v(s)$, to execute segment s and the expected execution time for s and the remainder of the process are

$$v(s) = \frac{e}{I(s)} \qquad \mathrm{Ex}(t(s)) = \frac{I(s)^2}{e}, \tag{6}$$

where e is the remaining energy budget when s is executed. Thus, the minimum expected execution time for the whole process, while honoring the energy budget, is $I(\sigma)^2/E$.

The first thing to note is that each terminal segment, s, will execute with a constant voltage because the number of cycles $c_{\mathrm{x}}(s)$ and the energy budget allocation are fixed. (See Section 2.1.) From $e = c_{\mathrm{x}}(s)v$, it follows that $v = e/c_{\mathrm{x}}(s) = e/I(s)$ and from $t = c_{\mathrm{x}}(s)/v$ it follows that $t = I(s)^2/e$; both in agreement with (6). This completes the base step for an induction to verify (6).

Now consider a segment s where (6) describes the optimal policy for all $r \in \theta(s)$. Let e be the remaining energy budget. Some of that budget, e_1, will be devoted to s and the remainder, $e - e_1$, will be used for the rest of the process execution. Thus, the expected time to execute s plus the remainder of the process is

$$\begin{aligned}
\mathrm{Ex}(t(s)) &= \frac{c_{\mathrm{x}}(s)^2}{e_1} + \sum_{r\in\theta(s)} p(s,r)\,\mathrm{Ex}(t(r)) \\
&= \frac{c_{\mathrm{x}}(s)^2}{e_1} + \sum_{r\in\theta(s)} p(s,r)\frac{I(r)^2}{e-e_1} \\
&= \frac{c_{\mathrm{x}}(s)^2}{e_1} + \frac{(I(s)-c_{\mathrm{x}}(s))^2}{e-e_1}
\end{aligned} \tag{7}$$

using the inductive assumption and a constant-voltage solution for s. The minimizing value of e_1 is found by solving $\frac{d\,\mathrm{Ex}(t(s))}{d\,e_1} = 0$ for e_1. The result is $e_1 = c_{\mathrm{x}}(s)e/I(s)$. Since $e_1 = c_{\mathrm{x}}(s)v(s)$, it follows that $v(s) = e/I(s)$ in agreement with (6). Now substitute the optimizing value of e_1 into (7) and simplify to show that $\mathrm{Ex}(t(s)) = I(s)^2/e$ and complete the induction.

3.2.2 Hard Time Bound

The problem is to find a voltage schedule that minimizes expected energy utilization given the structural process model, $(\sigma, S, c_{\mathrm{x}}, p)$, and a not-to-exceed time budget T. The method to solve this problem is identical to the one used in Section 3.2.1. The results of this optimization are

$$v(s) = \frac{I(s)}{t} \qquad \mathrm{Ex}(e(s)) = \frac{I(s)^2}{t}, \tag{8}$$

where $I(s)$ is defined by (5) and t is the remaining time budget when s is executed. The expected energy utilization for the entire process is $I(\sigma)^2/T$.

3.2.3 Algorithms for Structural Models

The optimal policies developed for the structural process model are straightforward to implement. Voltage is set when each $s \in S$ begins execution as a function of $I(s)$ and the remainder of the budgeted resource; how control arrives at s is not relevant. Figure 2 shows API routines that are called at the beginning of the execution of s with the parameter $I(s)$.

```
procedure set_voltage (I)
    if e ≤ 0 then error;
    voltage ← e/I;
end set_voltage
```

(a) Energy management API.

```
procedure set_voltage (I)
    if t ≤ 0 then error;
    voltage ← I/t;
end set_voltage
```

(b) Time management API.

Figure 2: APIs to support structural process model execution.

The API shown in Figure 2(a) is used when energy is budgeted and e is the remaining energy budget. Figure 2(b) shows the API when time is the constraint and t measures remaining time. The error checks account for possible prior effects of parameter misestimations.

In order to use the optimal policy, $I(s)$ for each $s \in S$ must be available at runtime. These values can be calculated offline and inserted, along with the API calls, by a power-aware compiler. The total execution time to calculate all $I(s)$ values is proportional to the size of the graph used to represent the structural model of the process as an examination of Figure 3 will show. The recursive algorithm is called with σ, the initial segment, and when it completes, `s.I` is set for each $s \in S$. The mark fields `s.mark` are initially `false`; they are used so that `s.I` is calculated exactly once for each $s \in S$.

3.3 Loops

Loop constructs are a fundamental control structure that must be accounted for by any serious process modelling technique. The possibility of variable numbers of iterations is one of the major contributors to nondeterminacy in process complexity. The simple probabilistic model captures and organizes this information when p is estimated. That model is certainly the easiest and most natural one to use when there are many loops with variable iteration counts or elaborate control structures composed of ill-nested forms. Since many systems that interact with the environment have such characteristics, their architects will often choose the

```
function I(s segment)
    declare a number;
    if ¬s.mark then {
        s.mark ← true;
        a ← 0;
        for r ∈ θ(s) {
            a ← a + prob(s, r) * I(r)^2 };
        s.I ← c_x(s) + √a };
    return s.I;
end I
```

Figure 3: Algorithm to calculate $I(s)$ for each $s \in S$.

simple probabilistic model.

In some cases the complexity of one portion of a process is statistically dependent on the complexity of another, e.g., the iteration counts of various loops are related. If the dependencies are strong, multiple instances of some process portions would be necessary in a structural model so that the branch probabilities could capture the dependencies. The system modeler may well choose to use a simple probabilistic model rather than deal with the additional complexity.

The structural process model also provides mechanisms to account for loops. If, for example, the iteration count is fixed, the loop is simply unrolled. If only the maximum count is known, the loop is still unrolled with appropriate probabilities attached to the continuation and exit branches of each iteration.

The case where the iteration limit is unknown in advance or is difficult to calculate is more problematic. Consider the canonical loop structure shown in Figure 4 where segment h is the head of the loop and segment b is the body. The header enters the body with

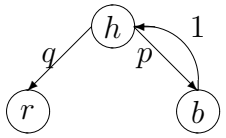


Figure 4: Simple loop structure.

probability p and segment r, the loop exit, with probability $q = 1 - p$. Two equations

immediately follow from (5):

$$I(h) = c_{\mathrm{x}}(h) + \sqrt{pI(b)^2 + qI(r)^2}$$
$$I(b) = c_{\mathrm{x}}(b) + I(h).$$

The simultaneous solutions for $I(h)$ and $I(b)$, with the assumption that p is independent of iteration count, are

$$I(h) = \frac{c_{\mathrm{x}}(h) + pc_{\mathrm{x}}(b) + \sqrt{p(c_{\mathrm{x}}(h) + c_{\mathrm{x}}(b))^2 + q^2 I(r)^2}}{q}$$
$$I(b) = \frac{c_{\mathrm{x}}(h) + c_{\mathrm{x}}(b) + \sqrt{p(c_{\mathrm{x}}(h) + c_{\mathrm{x}}(b))^2 + q^2 I(r)^2}}{q}.$$

The same technique—expand (5) for each node in the loop plexus then simultaneously solve for the I values—can be used to calculate metrics for more complicated cases including multiple and/or nested loops. The I values will be passed to the API (Section 3.2.3) to control voltage settings. Later iterations will see less of the budgeted resource remaining so the processor speed will rise or fall, appropriately, as execution continues.

Shin et al [41] require that the maximum iteration counts be known at compile time. When a loop exits before that count is reached, the worst-case expectation for the remaining process complexity is reduced. Zhu et al [48] propose a more sophisticated method of dealing with loops. The loop is unrolled but, depending on inter-iteration dependencies, the individual iterations can be represented as either serial or parallel tasks or some combination thereof. Probabilities are used to specify the likelihoods of actually executing each task cluster. Neither approach deals with the case where the bound on the number of iterations is a priori unknown.

4 Optimizations with General Power Models

The optimizations presented in Section 3 were all developed using the Π_{11} power model. Below, the results of these optimizations using the general Π_{mn} power models described in Section 2.1 are provided. The generalized results are grouped by the process model that is used: first the optimizations for the simple probabilistic model then those for the structural model. The techniques to derive these more general results are virtually identical to those assuming Π_{11} so they are not repeated.

4.1 Simple Probabilistic Model and Π_{mn}

The simple probabilistic process model posits the existence of a function p, where $p(x)$ is the probability that process complexity is exactly x cycles. The Π_{mn} power model states that $E(x) = \int_0^x v(y)^m \, dy$ and $T(x) = \int_0^x v(y)^{-n} \, dy$, where $E(x)$ is the total energy consumed executing cycles $0 \ldots x$, $T(x)$ is the total time to execute these cycles, and $v(y)$ is the voltage used at cycle y.

The optimization of Section 3.1.1 finds the function, $v(x)$, that achieves the least expected execution time given a not-to-exceed energy budget E using Π_{11}. The results of that optimization are captured in (2). When the Π_{mn} power model is used instead, the corresponding results are

$$v(x) = z(x)^{\frac{1}{m+n}} \left[\frac{E}{\Phi(m)}\right]^{\frac{1}{m}} \quad \mathrm{Ex}(T) = \frac{\Phi(m)^{\frac{m+n}{m}}}{E^{\frac{n}{m}}} \quad \mathrm{Ex}(E) = \frac{\Phi(2m+n)}{\Phi(m)} E, \tag{2'}$$

where

$$\Phi(r) = \int_0^c z(x)^{\frac{r}{m+n}} \, dx$$

is a non-increasing function of r.

The minimization of Section 3.1.2 is similar except that there is a not-to-exceed time budget, T, and the objective is to minimize expected energy consumption. The results of the optimization, corresponding to (3), using the Π_{mn} power model are

$$v(x) = \frac{1}{z(x)^{\frac{1}{m+n}}} \left[\frac{\Phi(n)}{T}\right]^{\frac{1}{n}} \quad \mathrm{Ex}(E) = \frac{\Phi(n)^{\frac{m+n}{n}}}{T^{\frac{m}{n}}} \quad \mathrm{Ex}(T) = \frac{\Phi(m+2n)}{\Phi(n)} T, \tag{3'}$$

where Φ is as defined above.

The optimization of Section 3.1.4 seeks to minimize the expected value of the penalty function $Q(E, T)$. The condition for a stationary solution, corresponding to (4), using the Π_{mn} power model, is

$$v(x)^{m+n} = \frac{n \int_x^c p(y) Q_2(y) \, dy}{m \int_x^c p(y) Q_1(y) \, dy}. \tag{4'}$$

Energy delay metrics of the form $Q(E, T) = E^\alpha T^\beta$, often used to measure architecture efficiency [25, 31, 49], are anomalous when used as an application metric. When one checks for a constant voltage stationary solution using that formula, v terms cancel and $m\alpha = n\beta$ is the residual. When the equality is true, *any* constant voltage produces the same expected value of Q; when the equality is false, either the maximum or minimum possible voltage will minimize Q.

4.2 Structural Process Model and Π_{mn}

Section 3.2 developed optimizations given a $(\sigma, S, c_{\mathrm{x}}, p)$ structural process model using the Π_{11} power model. This section presents the results of those optimizations when an arbitrary Π_{mn} power model is used instead.

Section 3.2.1 finds the optimal voltage to use for each $s \in S$ and the expected execution time when there is a not-to-exceed energy budget. The results analogous to (6) when the Π_{mn} power model is used are

$$v(s) = \left[\frac{e}{I(s)}\right]^{\frac{1}{m}} \qquad \mathrm{Ex}(t(s)) = \frac{I(s)^{\frac{m+n}{m}}}{e^{\frac{n}{m}}} \tag{6'}$$

where e is the energy remaining when s is executed and

$$I(s) = c_{\mathrm{x}}(s) + \Big(\sum_{r \in \theta(s)} p(s,r) I(r)^{\frac{m+n}{m}} \Big)^{\frac{m}{m+n}}. \tag{9}$$

Section 3.2.2 finds the optimal voltage to use for each $s \in S$ and the expected energy utilization when there is a not-to-exceed time budget. The results analogous to (8) when the Π_{mn} power is used are

$$v(s) = \left[\frac{I(s)}{t}\right]^{\frac{1}{n}} \qquad \mathrm{Ex}(e(s)) = \frac{I(s)^{\frac{m+n}{n}}}{t^{\frac{m}{n}}} \tag{8'}$$

where t is the time remaining when s is executed and

$$I(s) = c_{\mathrm{x}}(s) + \Big(\sum_{r \in \theta(s)} p(s,r) I(r)^{\frac{m+n}{n}} \Big)^{\frac{n}{m+n}}.$$

Note that forms such as $(\sum \alpha_i x_i^r)^{1/r}$ are known as general weighted (or Hölder) means and are increasing functions of r [19].

5 Metric Estimates and Deployment

Implementing the optimizations developed above requires metrics about the hardware platform to determine the proper power model to use and metrics about the application to model it and compute an optimal voltage schedule. Hardware performance metrics and tradeoffs are typically documented by chip vendors and can be supplemented with simulation and testing.

Information about applications is more specific and must be developed on a case-by-case basis. Pouwelse and Langendoen [37] argues that voltage scaling can only be effective when applications cooperate and Barnett [5] describes how applications form energy-related tradeoffs to measure and enhance system performance. Sources of applications metrics include developer intuition and domain knowledge, simulation, profiling, and feedback from the executing systems. Section 6 discusses the impact of errors in estimating these metrics. The remainder of this section briefly discusses methods to collect application-specific information and embed it in a system.

The most straightforward way to predict the execution time of a task is to gather execution times of previous instances of the same task [26]. For example, [48] assumes that probabilities and complexities not known a priori are determined by profiling. Lorch and Smith [29] discuss and analysis sophisticated methods to gather probabilities for the simple process model. The tradeoffs of using recent versus long-term statistics are explored along with how best to model the data collected, i.e., for what sort of distribution—normal, Pareto, etc.—should the parameters be estimated. It is argued that the goal of data collection and deployment is primarily to optimize system performance, not necessarily to capture the best representation of the distribution. The fact that the probabilities may not be stationary and could change over time as well as the difficulties of estimating meta-distributions are also noted. Gruian [17] analyzes performance differences between using a priori probability estimates offline and dynamically deriving estimates online and establishes guidelines for when one scheme works better than the other. Barnett [6] discusses the value and costs of using meta-distributions at compile and runtime in a general problem-solving context.

Once the application metrics have been gathered and the proper power model selected, there remains the problem of inserting power management points in the application. Designing, developing, and maintaining realtime code or any code that directly interacts with hardware devices are expensive error-prone activities [8]. Thus, approaches that shield the majority of the developers from the details of power and time management are needed to reduce costs and implement more robust power-aware systems. There are many approaches to technology insertion.

Some methods do optimizations and insertion offline [46] or place the entire burden for power management, online, in the operating system [16] or the scheduler [29]. Other approaches rely on power-aware compilers [17, 27, 34, 41] which certainly seems best when

feasible. Many compilers already use hardware models to count cycles and are aware, through dataflow analysis, of when future branch decisions are actually determined. This enables use of information about future process complexity as early as possible. Section 6.2 estimates the value of using such information promptly and the cost if there is a delay. It should also be possible to form power-aware versions of programming languages where pragmas are used to specify crucial application characteristics such as branch probabilities and sources of energy and time budget computations.

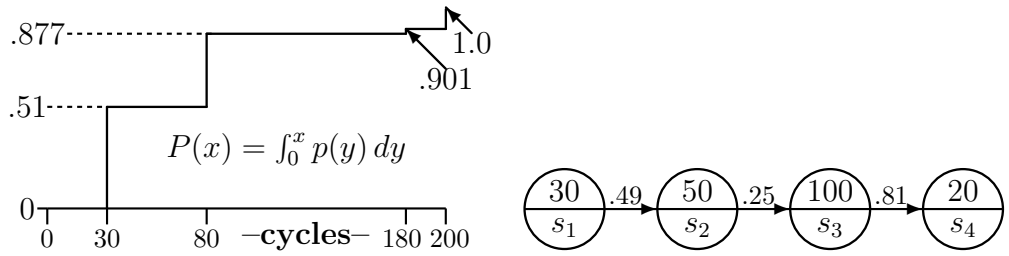
Assuming the power management mechanism is not buried beneath the application, the developers must still decide how often and where power management points appear in code, e.g, as calls on a power management API. (See Section 3.2.3.) The simplest approach is to adjust voltage at the release of each code segment or task [3, 16, 30, 41, 45, 48]. Lorch and Smith [29] discuss methods to break a task into chunks—each initiated by a separate power management point—using the simple probability model function, p, to guide the segmentation. It is noted that breaking tasks into ten segments, each scheduled at a constant voltage, is sufficient to get within 1.2% of an optimal energy solution where agile voltage control is assumed. Since it is not possible to vary voltage continuously or even for each cycle without incurring substantial overhead, methods such as these are extremely valuable. It is also straightforward to replace the integrals in (1) with summations over segments and optimize accordingly [16, 29]. Whatever strategy is used to place power management points in code, automated or semiautomated tools should be considered to easy the burden on application developers.

6 Analysis of Results

The formulas derived for optimal voltage scheduling are analyzed in this section. Section 6.1 compares the behavior of optimal schedules with schedules derived by considering average-case behavior. Section 6.2 analyzes the benefits of using information about future complexity promptly. Section 6.3 analyzes the impact on optimal performance when model parameters are misestimated. Finally, Section 6.4 develops theoretical criteria for when voltages change in an optimal schedule and when they should remain constant.

6.1 Estimates are Conservative

Figures 5(a) and 5(b) show, respectively, a simple probabilistic model represented by $P(x) = \int_0^x p(y)\,dy$ and a structural model for the same process. That process consists of the segments



(a) Simple probabilistic model (b) Structural model.

	s_1	s_2	s_3	s_4	max	Ex
$A(s_i)$	68.734	79.050	116.200	20.000		
$v(s_i)$	1.455	0.713	0.178	0.144		
$e(s_i)$	43.646	35.644	17.822	2.887	100.000	63.582
$t(s_i)$	20.620	70.137	561.098	138.543	790.398	137.469
$I(s_i)$	106.300	109.000	118.000	20.000		
$v(s_i)$	0.941	0.659	0.329	0.296		
$e(s_i)$	28.222	32.926	32.926	5.927	100.000	48.977
$t(s_i)$	31.890	75.929	303.714	67.492	479.025	112.997

(c) Metrics for average case and optimal strategies with $E = 100$.

Figure 5: Comparison of average case and optimal strategies.

$s_1, \ldots, s_4$. Each is executed in turn and the process may terminate after any of the segments. Optimal scheduling is compared, using this process example, with the intuitive idea of using average-case estimates of the remaining computation to set voltage levels [32, 48]. For simplicity, the Π_{11} model is assumed.

The average expected complexity of the process starting at s_i is $A(s_i) = c_{\mathrm{x}}(s_i) + p(s_i, s_{i+1})A(s_{i+1})$, where $A(s_4) = c_{\mathrm{x}}(s_4)$. If the problem were to reduce expected execution time within a total energy budget, E, the voltage used at s_i would be $v(s_i) = e_i/A(s_i)$, where e_i is the energy remaining just before s_i is executed and $e_1 = E$. The optimal strategy is $v(s_i) = e_i/I(s_i)$, where $I(s_i) = c_{\mathrm{x}}(s_i) + p(s_i, s_{i+1})^{1/2} I(s_{i+1})$ and $I(s_4) = c_{\mathrm{x}}(s_4)$. (This is just (5) specialized to the example process.) Since $A(s_i) < I(s_i)$ except for the trivial

cases where $p = 1$ or $p = 0$, the voltage selected by the average-case strategy at s_i will be greater given the same e_i. However, the energy budget remaining if s_{i+1} is executed will be less. Thus one may be motivated to describe the optimal strategy as conservative in that it reserves more energy for future contingent executions. Note that 1) these observations follow from (9) so are valid for *all* power models and 2) $A(s_i) < I(s_i)$ remains valid for more complicated structural models too.

Figure 5(c) shows metrics associated with the execution of each s_i as well as the expected total execution time and energy expenditure. The rows labelled $e(s_i)$ and $t(s_i)$ report, respectively, the energy needed to execute s_i and the time it would take. Maximum expected total energy and time appear in the columns labelled, respectively, max and Ex. The maximum energy used by both strategies is $E = 100$ when all segments execute. However, the expected energy expenditure is 23.0% less if the optimal strategy is used. When delays are compared, the optimal strategy reduces the maximum execution time by 39.4% and the expected execution time by 17.8%.

6.2 Promptness is a Virtue

It was noted in Section 2.2.3 that the structural model generally provides more information than a simple probabilistic model for the same process. The reason stated was that information about future behavior would be available sooner so that more informed voltage scheduling decisions could be made. An example is used to illustrate the value of prompt information deployment.

Consider a process whose initial segment complexity is $2a$ with a second segment of complexity ra executed with probability p. The expected time to execute this process, assuming the Π_{11} model, is I^2/E, where $I = 2a + p^{1/2}ra$ and E is the total energy budget. What is the effect if the decision to execute the second segment or terminate were known after a cycles? In this case, a more informed model would posit an initial segment of complexity a with a branch to one of two final segments: one segment, with complexity $a + ra$, would be executed with probability p and the second, with complexity a, would be executed with probability $q = 1 - p$. Here, the expected process execution time is J^2/E, where $J = a + (p(a + ra)^2 + qa^2)^{1/2}$. It is easy to show that $J < I$ unless $p = 0$, $p = 1$, or $r = 0$.

The question is what fraction of the expected execution time does early information

availability and use save? The fraction of time saved is simply

$$\frac{I^2/E - J^2/E}{I^2/E} = 1 - \left(\frac{1 + \sqrt{p(1+r)^2 + q}}{2 + \sqrt{p}\, r} \right)^2 .$$

Figure 6 shows this fraction for several values of p as a function of r. The maximum potential savings increase as p becomes smaller, however, the r where the maximum occurs becomes

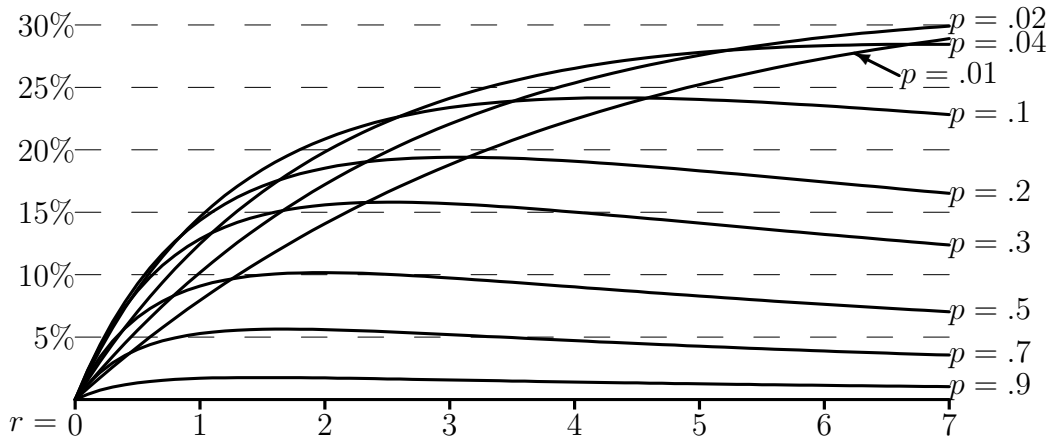


Figure 6: Fraction of execution time saved by prompt information use.

larger. The reason is that extra energy is committed to the second a-cycle segment, but in the rare case where the segment of complexity ra is executed, the penalty is rather large. Similar findings follow when general Π_{mn} models are used and when time rather than energy is budgeted. The minimization process is accelerated when new information is used promptly. While it may be difficult to manually optimize code to use information as soon as possible, it is the sort of task that could be performed by a power-aware compiler through dataflow and branch analysis.

Gruian [17] notes that in principle it is good to execute segments with the largest uncertainties earliest. This insight means that the code segments with the largest complexity variance, as determined by its distribution, should be executed first when execution order is optional. The concept of generating and using information as soon as possible to reduce expected costs is in agreement with the analysis herein.

6.3 Robustness of Results

Estimates of various probabilities and execution complexity are based on approximate models and limited testing so might not be exact. (See Section 5.) Further, the Π_{mn} power

models ignore threshold voltage, dynamic states of caches and pipelines, and environmental influences all of which effect the energy vs time tradeoff. This section investigates some of these sources of inaccuracies and their impacts on claims of optimality for the voltage scheduling techniques developed above. While these techniques will be shown to be robust, one must always be cautious when applying theory to practical cases that are not exactly modelled. The Π_{11} model will be used to simplify the analyses but note that the form of the results and qualitative assertions carry over to the general case.

6.3.1 Simple Distribution Estimation Errors

Assume that $p(x)$ is an accurate simple probabilistic model and a fixed energy budget E is given. Then from (2), we know that $\mathrm{Ex}(T) = I^2/E$ and $\mathrm{Ex}(E) = E\int_0^c z(x)^{2/3}dx/I$. The issue to be investigated here is the effect on $\mathrm{Ex}(T)$ and $\mathrm{Ex}(E)$ if some $p_1 \neq p$ were used, instead of p, to determine the voltage schedule. That voltage schedule would be $v_1(x) = Ez_1(x)^{1/2}/I_1$, where $z_1(x) = 1 - \int_0^x p_1(y)\,dy$ and $I_1 = \int_0^c z_1(x)^{1/2}dx$. Therefore, the expected resource utilizations are

$$\begin{aligned}\mathrm{Ex}(E_1) &= \int_0^c z(x)v_1(x)\,dx & \mathrm{Ex}(T_1) &= \int_0^c z(x)/v_1(x)\,dx\\ &= \frac{E}{I_1}\int_0^c z(x)z_1(x)^{1/2} & &= \frac{I_1}{E}\int_0^c \frac{z(x)}{z_1(x)^{1/2}}dx\end{aligned}$$

and the measures we seek are

$$\mathrm{Er}(E) = \frac{\mathrm{Ex}(E_1) - \mathrm{Ex}(E)}{\mathrm{Ex}(E)} \qquad \mathrm{Er}(T) = \frac{\mathrm{Ex}(T_1) - \mathrm{Ex}(T)}{\mathrm{Ex}(T)},$$

which are the fractions of expected additional energy and time consumed if p_1 is used instead of the true density function p. Of course these measures can be large if p and p_1 are substantially different.

If p and p_1 are reasonably alike, e.g., if it is assumed that 1) $I = I_1$, 2) $|z_1(x)^{1/2} - z(x)^{1/2}| < \delta$, and 3) $z(x)^{1/2}/z_1(x)^{1/2} < 1 + \epsilon$, then bounds on $\mathrm{Er}(E)$ and $\mathrm{Er}(T)$ are straightforward to calculate: namely,

$$\mathrm{Er}(E) < \frac{\delta\,\mathrm{Ex}(c)}{\int_0^c z(x)^{3/2}dx} \qquad \mathrm{Er}(T) < \frac{\delta(1+\epsilon)c}{I},$$

where $\mathrm{Ex}(c) = \int_0^c z(x)\,dx$ is the expected process complexity. Both bounds are reasonable small if ϵ and δ are small and p does not have an extremely long, low-valued tail. It is interesting to compare this conclusion with the discussion in Section 6.2 of worst case behavior exhibited in Figure 6.

6.3.2 Structural Model Errors

Executions based on the structural process model are robust in that they optimally recover from estimation errors. That robustness is serendipity form the API algorithm shown in Figure 2. Even if earlier execution has been based on misestimated c_{x} and p values, the optimal continuation is to execute segment s with voltage $e/I(s)$ or $I(s)/t$, when $I(s)$ is correctly estimated, because e, respectively t, is the actual remaining resource. If, however, e, respectively t, were projected by offline analysis, the prior errors and divergence from optimality would be exacerbated.

The remainder of this section considers the effect when a set of branch probabilities are misestimated and the objective is to minimize expected execution time given an energy budget. The Π_{11} power model will be assumed for simplicity, but note that similar results are available for the general Π_{mn} model and the problem where time is the constraint.

Consider the situation just before segment s is executed with remaining energy e. The total expected remaining execution time, from (6), is $\mathrm{Ex}(T) = I(s)^2/e$, where $I(s) = c_{\mathrm{x}}(s)+\psi$ and $\psi = \left(\sum_{x\in\theta(s)} p(s,x)I(x)^2\right)^{1/2}$. Assume that the $p(s,x)$ are misestimated so that $I_1(s) = c_{\mathrm{x}}(s) + \psi_1$ is believed instead of the true value. Thus, s will be executed with voltage $v_1 = e/I_1(s)$ and consume energy $e_1 = ec_{\mathrm{x}}(s)/I_1(s)$ leaving energy $e_2 = e - e_1 = e\psi_1/I_1(s)$ for the remaining execution. Executing s will take time $t_1 = c_{\mathrm{x}}(s)/v_1 = c_{\mathrm{x}}(s)I_1(s)/e$.

Since the $I(x)$, where $x \in \theta(s)$, are assumed correct—only the p values are suspect—the actual expected remaining execution time is $t_2 = \psi^2/e_2 = I_1(s)\psi^2/(e\psi_1)$. Therefore, the actual expected total execution time with the false assumption is

$$\begin{aligned}\mathrm{Ex}(T_1) &= t_1 + t_2 \\ &= \frac{(c_{\mathrm{x}}(s)\psi_1 + \psi^2)(c_{\mathrm{x}}(s) + \psi_1)}{\psi_1 e}.\end{aligned}$$

The relative loss from the false assumption is

$$\mathrm{Er}(T) = \frac{\mathrm{Ex}(T_1) - \mathrm{Ex}(T)}{\mathrm{Ex}(T)}.$$

Now define α and β such that $\psi = \alpha c_{\mathrm{x}}(s)$ and $\psi_1 = \beta c_{\mathrm{x}}(s)$ and substitute in the above to show that

$$\mathrm{Er}(T) = \frac{(\alpha - \beta)^2}{\beta(1 + \alpha)^2}.$$

The objective is to analyze $\mathrm{Er}(T)$ in terms of the estimation error. Define $r = \alpha/\beta$ and rewrite the above as

$$\mathrm{Er}(T) = \frac{\beta(r-1)^2}{(1+r\beta)^2}.$$

If $|r-1| < \epsilon$, for some $\epsilon > 0$,

$$\mathrm{Er}(T) < \frac{\beta\epsilon^2}{(1+r\beta)^2} \leq \frac{\epsilon^2}{4r},$$

where the second inequality follows because the maximum value of $\beta/(1+r\beta)^2$ occurs when $\beta = r^{-1}$. Thus, if ϵ is small so is $\mathrm{Er}(T)$.

6.4 When Can/Should Voltage Change?

A simple an important theoretical result follows immediately from the developments in Section 4.1. Voltage should only change in an optimal voltage schedule when executing in a region where the probability of termination is nonzero. Table 3 summarizes the voltage

min	$\mathrm{Ex}(T)$	$\mathrm{Ex}(E)$	$\mathrm{Ex}(Q(E,T))$
$v(x) =$	$z(x)^{\frac{1}{m+n}}\left[\dfrac{E}{\Phi(m)}\right]^{\frac{1}{m}}$	$\dfrac{1}{z(x)^{\frac{1}{m+n}}}\left[\dfrac{\Phi(n)}{T}\right]^{\frac{1}{n}}$	$\left[\dfrac{n\int_0^c p(y)Q_2(y)\,dy}{m\int_0^c p(y)Q_1(y)\,dy}\right]^{\frac{1}{m+n}}$

Table 3: Summary of agile voltage scheduling results.

computations for simple probabilistic models where agile voltages—those that can change anywhere—are considered. Assume that $p(x) = 0$ when $x_1 \leq x \leq x_2$. Then $z(a) = z(b)$ for any $x_1 \leq a, b \leq x_2$ because $z(x) = 1 - \int_0^x p(y)\,dy$. Therefore, the result is immediate when the objective is to minimize $\mathrm{Ex}(E)$ or $\mathrm{Ex}(T)$. It is also immediate when the objective is to minimize $\mathrm{Ex}(Q)$ because $\int_a^c p(y)Q_i dy = \int_b^c p(y)Q_i dy$ when p, a, and b have the assumed properties. Another way to summarize this fact is that voltage will remain constant unless something new is learned; what can be learned during the execution of cycle x is that the process did or did not terminate at cycle x. Such a discrimination is only possible when $p(x) \neq 0$.

Similar observations follow when optimal voltage schedules are derived using a structural process model. Voltage never changes during the execution of a segment. However, when new information becomes available about process complexity by choosing a branch, the voltage level might change. So the general conclusion, using either model, is that voltage can only

change in an optimal schedule when something new is learned about the future complexity of the process.

7 Conclusions

The analysis and form of the optimal policies developed above show that less of the budgeted resources—energy or time—is allocated initially than would be used if one assumed average-case behavior for future process complexity. Such aggressive non-optimal scheduling policies will pay a substantial penalty in instances where the process complexity is relatively large, particularly if such cases are rare because the computation of the average does not fully account for their nonlinear effect on expected resource utilization. The gain that aggressive policies enjoy most of the time will be more than offset and the expected value of the parameter to be minimized will be greater.

Adequate testing must be done to fully understand these effects when non-analytic, heuristic optimizations are used. The test set must be large enough to include a fair share of outliers. For example, if maximum process complexity is as much as 3σ (σ is standard deviation) more than the average complexity, a thousand or more test cases would be necessary to properly benchmark expected scheduler performance. Of course if a system will execute only a limited number of times or the optimization objective is other than minimizing expected value of resource consumption, less exhausting testing procedures may be adequate. The analytic results developed herein can be used to provide bounds on expected resource consumption as well as indicate significant deviations between a proposed scheduling policy and the optimal one. This information would be used to parametrically tune and improve specialized scheduling methods developed for particular applications.

The optimal analytic results can also be used to estimate the value of more testing to determine model parameters. For example, the p function of the simple probabilistic model will usually be a discrete rather than a continuous function because limited testing data is naturally organized into intervals and because it is rather unlikely that an exact distribution can be inferred from the process code or limited testing. Typically, the complexity range, $0 \ldots c$, will be partitioned into intervals, $c_i \ldots c_{i+1}$, and a termination probability p_i estimated for each one. The potential gain in collecting more data is approximately measured by comparing the expected resource consumption under the assumption that the mass of p_i is

concentrated at the beginning and under the assumption it is concentrated at the end of the interval. These extreme allocations of the probability mass represent the maximum and minimum possible future complexities consistent with the data. If the difference is small enough to satisfy applications desiderata, then little will be gained by further data collection and analysis.

Discrete representation of the probability distribution endows the simple model with a property inherent in the structural model: a collection of segments that each will execute at a constant voltage. Constant voltage executions simplify practical problems that must be accounted such as discrete voltage levels, maximum and minimum voltage levels, and resource penalties for switching voltage level and/or clock speed. Incorporating these limitations into scheduling disciplines will assist the architects of power-aware systems in choosing the best size of segments and determining how much statistical testing is necessary to achieve acceptable system performance. The results developed herein will provide theoretical boundaries and limits on the possibilities of such efforts.

Acknowledgements: This effort is sponsored by Defense Advanced Research Projects Agency (DARPA) through the Air Force Research Laboratory, USAF, under agreement number F33615-02-C-4001.

References

[1] *Advanced Micro Devices Corporation*, http://www.amd.com, March 2004.

[2] AeroVironment Corporation, http://www.aerovironment.com, March 2004.

[3] H. Aydin, R. Melhem, D. Mossé, P. Mejía-Alvarez, Determining optimal processor speeds for periodic real-time tasks with different power characteristics, *Euromicro Conference on Real-Time Systems*, pp. 225–232, 2001.

[4] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez, Optimal reward-based scheduling for periodic real-time tasks, *IEEE Transactions on Computers*, 50(2), pp. 111–130, 2001.

[5] J. Barnett, Application-level power awareness, In R. Graybill and R. Melhem (Eds.), *Power Aware Computing*, Kluwer, pp. 227-242, 2002.

[6] J. Barnett, How much is control knowledge worth? A primitive example, *Artificial Intelligence*, 22(1), pp. 77–89, 1984.

[7] K. Błażewicz, E. Ecker, E. Pesch, G. Schmidt, and J. Węglarz, *Scheduling Computer and Manufacturing Processes*, Springer-Verlag, Berlin, Germany, pp. 346–350, 1996.

[8] B. Bohem, E. Horowitz, R. Madachy, D. Reifer, B. Clark, B. Steece, A. Brown, S. Chulain, C. Abts, Software Cost Estimates with COCOMO II, Prentice Hall, 2000.

[9] R. Buck, *Advanced Calculus*, McGraw Hill, pp. 296–298 and 375–385, 1956.

[10] T. Burd and R. Brodersen, Energy efficient CMOS microprocessor design, *HICSS Conference*, pp. 288–297, 1995.

[11] T. Burd, T. Pering, A. Statakos, and R. Brodersen, A dynamic voltage scaled microprocessor systems, *IEEE Journal of Solid-State Circuits*, 35(11), pp. 1571–1580, 2000.

[12] G. Cao, Proactive power-aware cache management for mobile computing, *IEEE Transactions on Computers*, 51(6), pp. 608–621, 2002.

[13] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, Data driven signal processing: An approach for energy efficient computing, *ISLPED*, pp. 347–352, 1996.

[14] A. Chandrakasan, S. Sheng, and R. Brodersen, Low-power CMOS digital design, *IEEE Journal of Solid-State Circuits*, 27(4), pp. 473–484, 1992.

[15] D. Gillies and W. Liu, Scheduling tasks with AND/OR precedence constraints, *SIAM Journal of Computing*, 24(4), pp. 797–810, 1995.

[16] F. Gruian, Hard real-time scheduling for low-energy using stochastic data and DVS processors, *ISLPED*, pp. 46–51, 2001.

[17] F. Gruian, On energy reduction in hard real-time systems containing tasks with stochastic execution times, *IEEE Workshop for Real-Time Embedded Systems*, pp. 11–16, 2001.

[18] V. Gutnik and A. Chandrakasan, An efficient controller for variable supply voltage low power processing, *Symposium on VLSI Circuits* pp. 158–159, 1996.

[19] G. Hardy, J. Littlewood, and G. Pólya, *Inequalities*, Cambridge University Press, 1988.

[20] M. Igarashi, K. Usami, K. Nogami, F. Minami, Y. Kawasaki, T. Aoki, M. Takano, C. Mizuno, T. Ishikawa, M. Kanazawa, S. Sonoda, M. Ichida, and N. Hatanaka, A low-power design method using multiple supply voltages, *ISLPED*, pp. 36–41, 1997.

[21] *Intel Corporation*, http://developer.intel.com/design/intelxscale, March 2004.

[22] Intel, Microsoft, and Toshiba, *Advanced Configuration and Power Management Interface (ACPI) Specification*, http://www.acpi.info, March 2004.

[23] *International Business Machines*, http://www.ibm.com, March 2004.

[24] T. Ishihara and H. Yasuura, Voltage scheduling problems for dynamically variable voltage processors, *ISLPED*, pp. 197–202, 1998.

[25] J. Jalminger and P. Stenström, Improvement of energy-efficiency in off-chip caches by selective prefetching, *Microprocessors and Microsystems* 26(3), pp. 107–121, 2002.

[26] P. Kumar and M. Srivastava, Predictive strategies for low-power RTOS scheduling, *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Austin, TX, pp. 343, 2000.

[27] S. Lee, and T. Sakurai, Run-time voltage hopping for low-power real-time systems, *37th Design Automation Conference*, pp. 806–809, 2000.

[28] D. Li, P. Chou, and N. Bagherzadeh, Topology selection for energy minimization in embedded networks, *Asia South-Pacific Design Automation Conference (ASPDAC)*, pp. 693–696, 2003.

[29] J. Lorch and A. Smith, Improving dynamic voltage scaling algorithms with PACE, *ACM Sigmetrics*, pp. 50–61, 2001.

[30] A. Manzak and C. Chakrabarti, Variable voltage task scheduling algorithms for minimizing energy, *ISLPED*, pp. 279–282, 2001.

[31] A. Martin, M. Nyström, and P. Pénzes, Et^2: A metric for time and energy efficiency of computation, In R. Graybill and R. Melhem (Eds.), *Power Aware Computing*, Kluwer, pp. 293–315, 2002.

[32] R. Melhem, N. AbouGhazaleh, and D. Mossé, Power management points in power-aware real-time systems, In R. Graybill and R. Melhem (Eds.), *Power Aware Computing*, Kluwer, pp. 127–152, 2002.

[33] S. Mohanty, J. Ou, and V. Prasanna, An estimation and simulation framework for energy efficient design using platform FPGAs, *IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 290, April 2003.

[34] D. Mossé, H. Aydin, B. Childers, and R. Melhem, Compiler-assisted dynamic power-aware scheduling for realtime applications, *Workshop on Compilers and Operating Systems for Low Power*, October 2000.

[35] W. Namgoang, M. Yu, and T. Meg, A high efficiency variable-voltage CMOS dynamic DC-DC switching regulator, *IEEE International Solid-State Circuits Conference*, pp. 380–391, 1997.

[36] P. Pillai and K. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems, *ACM Symposium on Operating Systems Principles*, pp. 89–102, 2001.

[37] J. Pouwelse, K. Langendoen, H. Sips, Energy priority scheduling for variable voltage processors, *ISLPED*, pp. 28–33, 2001.

[38] Q. Qiu and M. Pedram, Dynamic power management based on continuous-time markov decision processes, *Design Automation Conference 36*, pp. 555–561, 1999.

[39] V. Raghunathan, P. Spanos, and M. Srivastava, Adaptive power-fidelity in energy aware wireless embedded systems, *IEEE Real-Time Systems Symposium*, pp. 106, 2001.

[40] T. Sakurai and A. Newton, Alpha-power law MOS-FET models and its applications to CMOS inverter delay and other formulas, *IEEE Journal of Solid-State Circuits*, 25(2), pp. 584–594, 1990.

[41] D. Shin, J. Kim, and S. Lee, Intra-task voltage scheduling for low-energy hard real-time applications, *IEEE Design and Test of Computers*, 18(2), pp. 20–30, March-April 2001.

[42] P. Shriver, M. Gokhale, S. Briles, D. Kang, M. Cai, K. McCabe, S. Crago, and J. Suh, A power-aware, satellite-based parallel signal processing scheme, In R. Graybill and R. Melhem (Eds.), *Power Aware Computing*, Kluwer, pp. 243-259, 2002.

[43] K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, and T. Kuroda, 300MIPS/W RISC core processor with variable voltage supply-voltage scheme in variable threshold-voltage CMOS, *Proceedings of the ICC*, pp. 587–590, 1997.

[44] *Transmeta Corporation*, http://www.transmeta.com, March 2004.

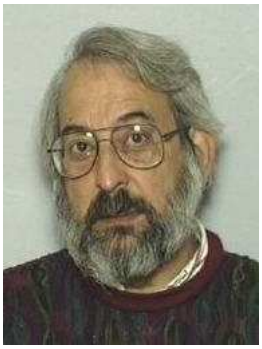
[45] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, Energy-aware runtime scheduling for embedded-multiprocessor SOCs, *IEEE Design & Test of Computers*, 18(5), pp. 46–58, 2001.

[46] F. Yao, A. Demers, and S. Shenker, A scheduling model for reduced CPU energy, *IEEE Annual Symposium on Foundations of Computer Science*, pp. 374–382, 1995.

[47] W. Ye, J. Heidemann, and D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, *IEEE Computer and Communications Societies (INFOCOM)*, pp. 1567–1576, 2002.

[48] D. Zhu, D. Mossé, and R. Melhem, Power aware scheduling for AND/OR graphs in real-time systems, IEEE Transactions on Parallel and Distributed Systems, 15(9), pp. 849–864, 2004.

[49] V. Zyuban, P. Kogge, Inherently lower-power high-performance superscalar architectures, *IEEE Transactions on Computers*, 50(3), pp. 268–285, 2001.



Author Biography

Mr. Barnett did his undergraduate work in mathematics at UCLA and Indian University and his graduate work in computer science at University of California in Irvine. His research interests include artificial intelligence, mathematics, and system and process architectures. He has been at the Northrop Grumman Corporation for twenty years where he was a cofounder of the Automation Sciences Laboratory. Prior to that time, he was on staff at USC/ISI and was visiting faculty at UCLA and University of California in Irvine. His interest in power-aware computing was motivated by the enthusiasm of his colleague Bill Athas.