A Scheduling Model for Reduced CPU Energy

Frances Yao Alan Demers Scott Shenker

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{yao, demers, shenker}@parc.xerox.com

(extended abstract)

Abstract

The energy usage of computer systems is becoming an important consideration, especially for battery-operated systems. Various methods for reducing energy consumption have been investigated, both at the circuit level and at the operating systems level. In this paper, we propose a simple model of job scheduling aimed at capturing some key aspects of energy minimization. In this model, each job is to be executed between its arrival time and deadline by a single processor with variable speed, under the assumption that energy usage per unit time, P, is a convex function of the processor speed s. We give an off-line algorithm that computes, for any set of jobs, a minimum-energy schedule. We then consider some on-line algorithms and their competitive performance for the power function $P(s) = s^p$ where $p \geq 2$. It is shown that one natural heuristic, called the Average Rate heuristic, uses at most a constant times the minimum energy required. The analysis involves bounding the largest eigenvalue in matrices of a special type.

1 Introduction

Computers are rapidly becoming more widespread and more portable. For portable computers running on batteries, energy conservation is critically important. In a typical laptop computer, energy use is dominated by the backlit display and the disk. It is difficult to modulate the power consumption of these devices while they are operating, so energy saving techniques primarily involve turning them off after a period of no use.

The new generation of very small portable computers (PDAs) often have no disk at all, and lack the backlight that consumes much of the display-related power. For such devices, the power consumption of the CPU itself becomes significant. This fact is important because there are energy conservation techniques for CPUs that do considerably better than simply turning off the device during its "idle loop". In particular, CPU circuitry can be designed so that slower clock speeds use lower supply voltage, thus resulting in lower energy consumption per instruction (see [1,2,4,7] for various approaches). Such variable speed processors can operate reliably over a range of clock speeds. The power (i.e., energy per unit time) consumed by such a processor is a convex function of its execution speed, with the exact form dependent on the details of the technology.

On a computer with a variable speed processor, the operating system can reduce the energy consumption by scheduling jobs appropriately. Scheduling to reduce power consumption was first discussed in [7], which described several scheduling heuristics and measured the energy savings on typical work loads. This work was later extended in [3].

In this paper, we provide a more formal analysis of the minimum-energy scheduling problem. We propose a simple model in which each job is to be executed between its arrival time and deadline by a single variable-speed processor as described above. A precise definition of the model is given in Section 2. In Section 3, we give an off-line algorithm that computes a minimum-energy schedule for any set of jobs, with no restriction on the power consumption function except convexity. We then consider on-line heuristics in Section 4, with special focus on what we call the Average Rate heuristic (AVR). In Section 5, we prove that AVR has a constant competitive ratio, i.e., it uses at most a constant times the minimum energy required,

0272-5428/95 $04.00 © 1995 IEEE

assuming a quadratic power function $P(s) = s^2$. Our analysis shows that the ratio lies between 4 and 8. In Section 6, we sketch a constant-ratio proof for the general case $P(s) = s^p$ where $p \geq 2$. There, the ratio is shown to be between p^p and $2^{p-1}p^p$. Finally, we close with a discussion of some simulation results and open problems.

2 The Model

Let $[t_0, t_1]$ be a fixed time interval. An instance of the scheduling problem is a set J of *jobs* to be executed during $[t_0, t_1]$. Associated with each job $j \in J$ are the following parameters:

- a_j its *arrival time*,
- b_j its *deadline* ($b_j > a_j$), and
- R_j its required number of CPU cycles.

We refer to $[a_j, b_j]$ as the *interval* of job j. A *schedule* is a pair $\mathcal{S} = (s, job)$ of functions defined over $[t_0, t_1]$:

- $s(t) \geq 0$ is the processor speed at time t;
- $job(t)$ defines the job being executed at time t (or *idle* if $s(t) = 0$).

discrete

We require that $s(t)$ and $job(t)$ be piecewise constant with finitely many discontinuities. A *feasible* schedule for an instance J is a schedule $\mathcal{S}$ that satisfies

$$\int_{a_j}^{b_j} s(t)\delta(job(t), j)dt = R_j$$

for all $j \in J$ (where $\delta(x,y)$ is 1 if $x = y$ and 0 otherwise). In other words, $\mathcal{S}$ must give each job j the required number of cyles between its arrival time and deadline (with perhaps intermittent execution). We assume that the power P, or energy consumed per unit time, is a convex function of the processor speed. The total energy consumed by a schedule S is[1]

$$E(\mathcal{S}) = \int_{t_0}^{t_1} P(s(t))dt.$$

The goal of the scheduling problem is to find, for any given problem instance, a feasible schedule that minimizes $E(\mathcal{S})$.

[1] In the remainder of this paper, unless otherwise specified, all integrals are taken with respect to t, with t_0 and t_1 as lower and upper limits. We will use abbreviated notations whenever possible.

3 The Minimum Energy Scheduler

In this section, we consider the off-line version of the scheduling problem. We first give a charaterization of an energy-optimal schedule for any set of n jobs, which then leads naturally to an $O(n \log^2 n)$ time algorithm for computing such schedules.

The characterization will be based on the notion of a *critical interval* for J, which is an interval in which a group of jobs must be scheduled at maximum, constant speed in any optimal schedule for J. The algorithm proceeds by identifying such a critical interval for J, scheduling those 'critical' jobs, then constructing a subproblem for the remaining jobs and solving it recursively. The optimal $s(t)$ is in fact unique, whereas $job(t)$ is not always so. The details are given below.

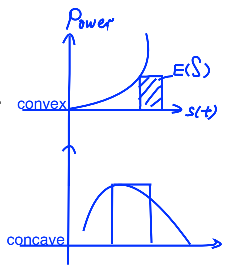
Definition. Define the *intensity* of an interval $I = [z, z']$ to be

$$① \quad \bar{s} \quad \geq g(I) = \frac{\sum R_j}{z' - z}$$

where the sum is taken over all jobs j with $[a_j, b_j] \subseteq [z, z']$.

①

Clearly, $g(I)$ is a lower bound on the average processing speed, $\bar{s}=\int_z^{z'} s(t)dt/(z' - z)$, that must be achieved by any feasible schedule over the interval $[z, z']$. Thus, by convexity of the power function, a schedule using constant speed $g(I)$ on $[z, z']$ is necessarily optimal on that interval (in the sense that no other feasible schedule can use less power on that interval).



Definition. Let $I^* = [z, z']$ be an interval that maximizes $g(I)$. We call I^* a *critical interval* for J, and the set of jobs $J_{I^*} = \{ j \mid [a_j, b_j] \subseteq [z, z'] \}$ the *critical group* for J.

Note that we can assume $I^* = [a_i, b_j]$ for some i, j. The following theorem shows that a critical interval will determine a segment of the optimal schedule. We omit the proof here.

Theorem 1. *Let I^* be a critical interval for J. If S is an optimal schedule for J, then the maximum speed of $\mathcal{S}$ is $g(I^*)$, and $\mathcal{S}$ runs at that speed for the entire interval I^*.*

(Moreover, S must execute every job of J_{I^*} completely within I^*, and execute no other jobs during I^*.) Theorem 1 immediately leads to the following algorithm, which finds an optimal schedule for J by computing a sequence of critical intervals iteratively.

Algorithm [Optimal-Schedule]

Repeat the following steps until J is empty:

1. Identify a critical interval $I^* = [z, z']$ by computing $s = \max g(I)$, and schedule the jobs of J_{I^*} at speed s over interval I^* by the *earliest deadline* policy (which is always feasible, see [6]);

2. Modify the problem to reflect the deletion of jobs J_{I^*} and interval I^*. This involves:
   let $J \leftarrow J - J_{I^*}$;
   reset any deadline $b_j \leftarrow \overset{z}{\cancel{t}}$ if $b_j \in [z, z']$, and $b_j \leftarrow b_j - (z' - z)$ if $b_j \geq z'$;
   reset the arrival times similarly.

Note that, after each iteration, the intensity $g(I)$ of some intervals I may increase (because I has been 'compressed'), which affects the evaluation of $\max g(I)$ in the next round. A straightward implementation of the above algorithm requires $O(n^2)$ time for $|J| = n$. By using a suitable data structure, such as the *segment tree*, one can reduce the running time to $O(n \log^2 n)$. We will skip the implementation details here.

A job instance J and its corresponding optimal schedule $\mathcal{S}$ are shown in Figure 1. To keep the diagram simple, there is only one job in each critical group. Job j is executed at speed s_j over interval I_j^*, which we represent by a shaded rectangle with base I_j^* and height s_j. The original job j is shown as a rectangle with base I_j and height d_j. (These two rectangles coincide when s_j is a local maximum, such as the case for $j = 1, 3$ in the example.) Note that, by the way $\mathcal{S}$ is constructed, the interval I of any job belonging to the j-th critical group must satisfy

$$I \subseteq \bigcup_{i \leq j} I_i^*. \tag{1}$$

4 On-line Scheduling Heuristics

Obviously there is a large space of possible heuristics for the online version of the minimum-energy scheduling problem. We will mention two simple heuristics that appear natural:

• **Average Rate:** Associated with each job j is its *average-rate requirement* or *density*

$$d_j = \frac{R_j}{b_j - a_j}.$$

We define a corresponding step function $d_j(t) = d_j$ for $t \in [a_j, b_j]$, and $d_j(t) = 0$ elsewhere. At any time t, the *Average Rate Heuristic* (AVR) sets the processor speed at

$$s(t) = \sum_j d_j(t),$$

and use the earliest-deadline policy to choose among available jobs. It is easy to see that the strategy yields a feasible schedule.

• **Optimal Available:** After each arrival, recompute an optimal schedule for the problem instance consisting of the newly arrived job and the remaining portions of all other available jobs. (Thus, the recomputation is done for a set of jobs all having the same arrival time.)

In the remainder of this paper, we will focus on the AVR heuristic and analyze its competitive ratio. Since the competitive ratio depends on the precise form of $P(s)$, and because the competitive analysis is fairly complex, we first focus our attention on the case where $P(s) = s^2$. This represents the simplest nontrivial version of the energy minimization problem.

Given a problem instance J, let $\mathrm{OPT}(J)$ denote the energy cost of an optimal schedule, and let

$$\mathrm{AVR}(J) = \int \Big(\sum_j d_j(t)\Big)^2 dt \tag{2}$$

denote the cost of the heuristic schedule. The competitive ratio of the heuristic is defined to be, as usual, the least upper bound of $\mathrm{AVR}(J)/\mathrm{OPT}(J)$ over all J. We first look at how AVR performs in some specific cases. Let $[t_0, t_1] = [0, 1]$, and $|J| = n$ in the following examples.

Example 1. The ith job has interval $[0, 1/2^{i-1}]$. All jobs have density $d_i = 1/2$, except $d_n = 1$. (See Figure 2.)

The optimal schedule for this example has constant speed $s(t) = 1$, and executes the jobs in the order $J_n, \ldots, J_1$, with total energy cost 1. By evaluating Eq. 2, one finds that the energy used by AVR approaches 2 as $n \to \infty$, resulting in $\mathrm{AVR}(J)/\mathrm{OPT}(J) = 2$.

Example 2. The ith job has interval $[0, i/n]$, and density $d_i = (n/i)^e$ where $e \geq 1$. (See Figure 3.)

It can be verified that the jobs will form critical groups in the order $J_1, \ldots, J_n$. When $e = 1$, the optimal schedule has constant speed 1 and AVR has cost 2 as $n \to \infty$, giving a ratio of 2 again. With a careful analysis, one can prove that the ratio $\mathrm{AVR}(J)/\mathrm{OPT}(J)$ is maximized at 4 when e is chosen to be 3/2.

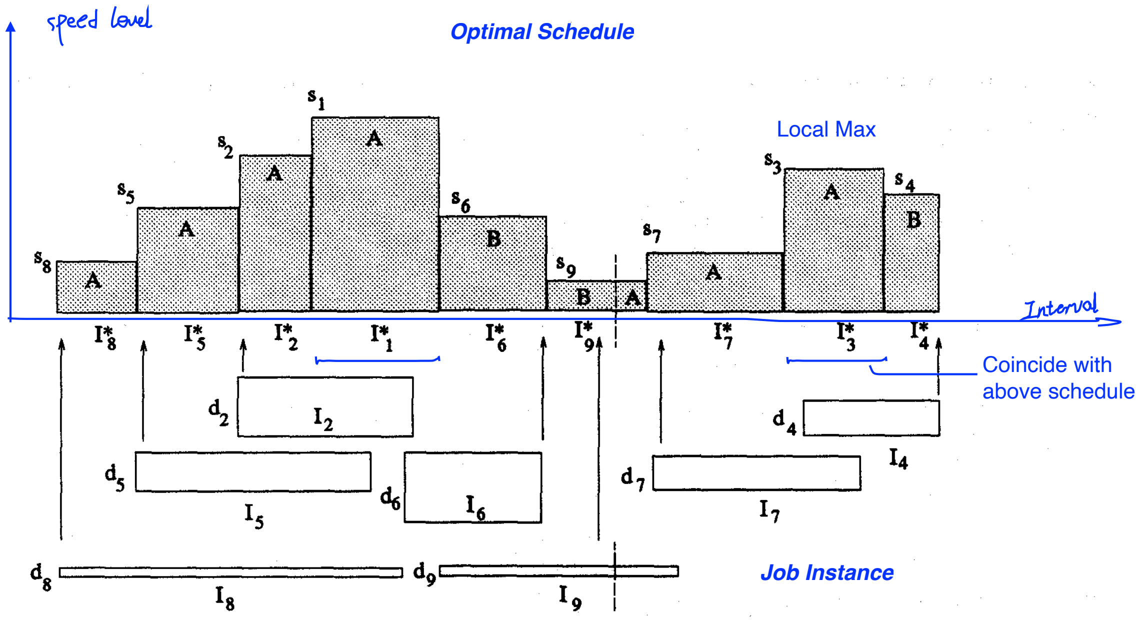


Figure 1: A job instance and its optimal schedule.

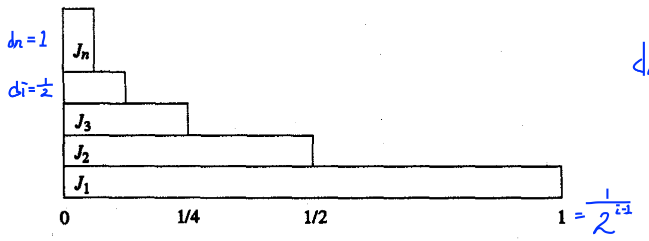


Figure 2: A set of jobs for which AVR has ratio 2.

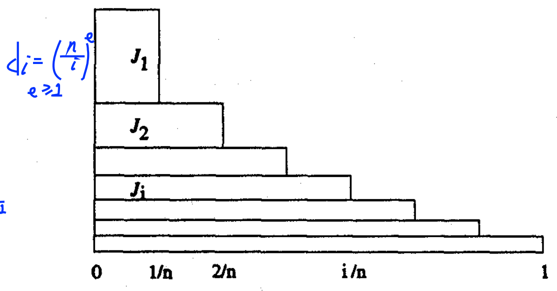


Figure 3: A set of jobs for which AVR has ratio 4.
$e = \frac{3}{2}$

Example 2, with $e = 3/2$, is the worst example we have for the AVR heuristic; we also conjecture that 4 is the exact value of its competitive ratio. In the next section, we prove a constant competitive ratio for the AVR heuristic. The constant we obtain is 4 for a restricted class of instances (which includes the preceding examples), and 8 for the general case.

5 Analysis of AVR

Throughout the analysis, we will assume that some speed function $s^*(t)$ is given, and consider only those job instances J which can be optimally scheduled by some $\mathcal{S} = (s^*(t), job(t))$ running at speed $s^*(t)$. We call $\langle J, \mathcal{S}\rangle$ a *candidate instance* for s^* (or simply an *instance* for s^*). We also refer to J as a candidate instance, if $\mathcal{S}$ is either unimportant or understood from the context.

We will carry out the analysis of AVR in two parts. In Section 5.1, we reduce the candidate instances to certain canonical forms. That is, we show that, for the purpose of obtaining an upper bound to AVR(J)/OPT(J), it suffices to consider candidate instances $\langle J, \mathcal{S}\rangle$ that satisfy certain special constraints. We then analyze the worst case ratio that is achievable under these constraints.

5.1 Canonical Forms

We first show that one can assume $\langle J, \mathcal{S}\rangle$ to be a "bitonic" instance; that is, the execution of each job $j \in J$ by $\mathcal{S}$ is either consistently "ahead of" the average rate, or consistently "behind" the average rate. Recall that, the function $s^*(t)\delta(job(t), j)$ specifies the execution speed of job j, which we will denote by $s^*_j(t)$. Thus,

$$\int_{a_j}^{b_j} s^*_j(t)dt = \int_{a_j}^{b_j} d_j(t)dt = R_j.$$

Let $\langle J, \mathcal{S}\rangle$ be a candidate instance for $s^*(t)$. We say $\langle J, \mathcal{S}\rangle$ is a *bitonic instance*, if every job $j \in J$ satisfies one of the following two inequalities:

$$\int_{a_j}^{t} s_j^*(t)dt \geq \int_{a_j}^{t} d_j(t)dt \quad \text{for } a_j \leq t \leq b_j, \quad (3)$$

or

$$\int_{a_j}^{t} s_j^*(t)dt \leq \int_{a_j}^{t} d_j(t)dt \quad \text{for } a_j \leq t \leq b_j. \quad (4)$$

We refer to j that satisfies Eq. 3 or Eq. 4 as a *type-A* or *type-B* job, respectively. (If j satisfies both inequalities, we assign its type arbitrarily.) If all jobs in J are of the same type, we call $\langle J, \mathcal{S}\rangle$ a *monotonic instance* for s^*.

We will show that, for the competitive analysis of AVR, the following restrictions may be placed on a candidate instance $\langle J, \mathcal{S}\rangle$:

1) [*bitonicity*] each job of J is either of type A or of type B;

2) [*non-preemption*] each job of J has a single execution interval;

3) [*alignment*] each I_j is aligned with execution-interval boundaries;

4) [*nesting*] the I_j's are properly nested.

A candidate instance $\langle J, \mathcal{S}\rangle$ satisfying the above restrictions is called a *canonical instance.*

Lemma 5.1 (Bitonicity) *Given an instance $\langle J, \mathcal{S}\rangle$ for s^*, there exists a bitonic candidate instance $\langle J', \mathcal{S}'\rangle$ for s^* such that AVR(J) =AVR(J').*

Proof. For a job $j \in J$, let $\{t_i\}$ be the points where the function $\Delta(t) = \int_{a_j}^{t}(s_j^*(t) - d_j(t))$ changes from nonzero to zero or vice versa. Split j into several new jobs (all having the same density as j) by dividing $[a_j, b_j]$ into subintervals at these points t_i. Modify $\mathcal{S}$ accordingly so that the resulting $\langle J', \mathcal{S}'\rangle$ is still a candidate instance for s^*. Clearly, $\langle J', \mathcal{S}'\rangle$ is bitonic and AVR(J) =AVR(J'). □

For the job instance shown in Figure 1, a single cut on J_9 transforms it into a bitonic instance. However, bitonicity does not preclude *preemptive exeuction*; that is, some job may have more than one execution intervals. This situation can make the analysis of AVR complicated. Therefore, we would like to further reduce the problem to non-preemptive cases only. For this purpose and, indeed, for the derivation of properties 3) and 4) below, we shall refer to a function $F(J)$ that closely approximates AVR(J) than to AVR(J) itself. We first motivate the definition of $F(J)$ with some preliminary discussions.

Write a bitonic instance as $J = J_A \cup J_B$, where J_A (respectively, J_B) consists of all the type-A (type-B) jobs. Our analysis will deal with the two subsets J_A and J_B separately and then combine the results. Define $s_A(t) = \sum_{i\in J_A} d_i(t)$, and $s_A^*(t) = \sum_{i\in J_A} s_i^*(t)$. Furthermore, let AVR$_A(J)$ and OPT$_A(J)$ denote the costs of AVR and OPT respectively that are attributable to J_A. That is,

$$\text{AVR}_A(J) \;=\; \int (s_A)^2 \quad \text{and} \quad \text{OPT}_A(J) = \int (s_A^*)^2.$$

Define s_B, s_B^*, AVR$_B$ and OPT$_B$ similarly. Then,

$$\begin{aligned} \text{OPT}(J) &= \text{OPT}_A(J) + \text{OPT}_B(J), \\ \text{AVR}(J) &\leq 2(\text{AVR}_A(J) + \text{AVR}_B(J)) \qquad (5) \end{aligned}$$

because of the inequality $(h+g)^2 \leq 2(h^2+g^2)$.

We will focus on the ratio of AVR$_A(J)$/OPT$_A(J)$ in the remainder of this section. Hence all jobs considered are assumed to be in J_A even without explicit mentioning. We first define a linear order for the jobs in J_A and relabel them as $J_1, J_2, \ldots$ accordingly . The linear order is consisitent with execution speed (i.e., the ordering of critical groups); hence $i < j$ if $s_i^* > s_j^*$. Among jobs in the same critical group, we order them by their arrival times a_j (which is equal to a_j^*): $i < j$ iff $a_i > a_j$. (For jobs in J_B, a linear order will be defined similarly, except that within the same critical group, we define $i < j$ iff $b_i < b_j$.) A useful property of such an ordering for J_A is the followling:

Lemma 5.2 *Let i, j be two jobs in J_A with $i < j$. Then $\int_{I_j} d_i \leq \int_{I_j} s_i^*$.*

Proof. The lemma is trivially true if $I_i \cap I_j = \emptyset$, hence we assume $I_i \cap I_j \neq \emptyset$. We claim that we must have $a_i > a_j$. If this were not true, then by the way the linear order is defined, i must belong to a higher-speed critical group than j. However, by Eq. 1, the entire interval of I_i would have been scheduled, making it impossible to execute job j right on arrival. Thus we have $a_i > a_j$. Now, the lemma follows from the type-A property of job i, in view of the fact that integration over I_j covers an initial portion of i's interval. □

Let $F_A(J) = 2\sum f_j$, where

$$f_j = \frac{R_j}{|I_j|}\sum_{i\leq j}\int_{I_j} s_i^*. \tag{6}$$

Then, it follows from Lemma 5.2 that

$$\begin{aligned}\mathrm{AVR}_A(J) \leq 2\sum_{i\leq j}\int d_i d_j &\leq 2\sum_j d_j(\sum_{i\leq j}\int_{I_j} s_i^*) \\ &= F_A(J). \end{aligned}\tag{7}$$

We now prove that, to maximize $F_A(J)$, we need only consider non-preemptive instances $\langle J, \mathcal{S}\rangle$ (that is, every job has a single execution interval). Incidentally, it is clear that the reduction in Lemma 5.1 satisfies $F_A(J) = F_A(J')$.

Lemma 5.3 (Non-preemption) *Given a bitonic instance $\langle J, \mathcal{S}\rangle$ for s^*, there exists a bitonic instance $\langle J', \mathcal{S}'\rangle$ for s^* such that $\langle J', \mathcal{S}'\rangle$ is non-preemptive and $F_A(J') \geq F_A(J)$.*

Proof. Suppose some job $j \in J$ has k disjoint execution intervals. We will show that j can be split into two jobs j' and j'', with $k-1$ and 1 execution intervals respectively, such that $f_j \leq f_{j'} + f_{j''}$ while s^* is unaffected. Non-preemption can then be achieved by induction on k. Let $I = [a, b]$ be the interval for j, and suppose its k-th execution interval starts at $t_0 \in [a, b]$. Write $I' = [a, t_0]$ and $I'' = [t_0, b]$. Let j' and j'' be two jobs with requirement $R' = \int_{I''} d_j^*$ and $R'' = \int_{I''} d_j^*$ respectively. Thus $R_j = R' + R''$. Let the interval of j'' be I''. To define the interval for j', we consider the average value of $s^*(t)$ over I' and over I''; that is, consider

$$\begin{aligned}\bar{s}^*(I') &= (\sum_{i\leq j}\int_{I'} s_i^*)/(t_0 - a) \qquad \text{and} \\ \bar{s}^*(I'') &= (\sum_{i\leq j}\int_{I''} s_i^*)/(b - t_0).\end{aligned}\tag{8}$$

We choose the interval of j' to be I if $\bar{s}^*(I') \leq \bar{s}^*(I'')$, and to be I' if $\bar{s}^*(I') > \bar{s}^*(I'')$. In the former case, the joint density of the two jobs $d'(t) + d''(t)$ is larger over I'' than over I', hence $f_{j'} + f_{j''} \geq f_j$. In the latter case, $d'(t)$, the joint density over I', is larger than $d''(t)$, the joint density over I'', because

$$\begin{aligned}\int_{I'} d' &= \int_{I'} d_j^* \geq \int_{I'} d_j, \\ \int_{I''} d'' &= \int_{I''} d_j^* \leq \int_{I''} d_j\end{aligned}$$

by the definition of type-A jobs. Hence $f_{j'} + f_{j''} \geq f_j$ is again true in this case. Finally, it is clear that the new job instance, with j replaced by $\{j', j''\}$, is still a candidate instance for $s^*(t)$. □

In a non-preemptive, bitonic instance for s^*, the execution interval $I_j^* = [a_j^*, b_j^*]$ of any job j occurs either at the beginning of $[a_j, b_j]$ (for type-A jobs), or at its end (for type-B jobs). In other words, if we specify a schedule $\mathcal{S} = (s^*(t), job(t))$ together with a bipartition $\gamma : \{1, 2, \ldots, n\} \mapsto \{A, B\}$, then any non-preemptive, bitonic instance J for s^*, whose partition $J = J_A \cup J_B$ coincides with γ, must satisfy the following:

1. *For job $j \in J$, its requirement is $R_j = \int_{I_j^*} s^*$.*
2. *For $j \in J_A$, its interval is of the form $[a_j^*, x_j]$ where $x_j \geq b_j^*$, and $a_1^* > a_2^* > \cdots$.*
3. *For $k \in J_B$, its interval is of the form $[y_k, b_k^*]$ where $y_k \leq a_k^*$, and $b_1^* < b_2^* < \cdots$.*

An instance J satisfying the above is said to be *consistent* with $(\mathcal{S}, \gamma)$. We would like to maximize $F_A(J)$ over all such instances for a given $(\mathcal{S}, \gamma)$. Note that, there are additional constraints (such as Eq. 1) which the variables x_j and y_k^* must satisfy, in order for J to be feasibly scheduled by $\mathcal{S}$. However, any upper bound derived for $F_A(J)$ under constraints 1)-3) only will certainly be a valid upper bound.

The next lemma shows that, in maximizing $F_A(J) = 2\sum f_j$, one can assume that each job interval I_j is aligned with execution-interval boundaries in $\mathcal{S}$. (In fact, the right endpoint x_j must coincide with b_k^* for some $k \in J_A$). We write $f_j(x_j)$ to indicate the dependency of f_j on x_j.

Lemma 5.4 (Alignment) *Among all instances consistent with $(\mathcal{S}, \gamma)$, the function F_A is maximized when each I_j, $j \in J_A$, is aligned with execution-interval boundaries in $\mathcal{S}$.*

Proof. Write

$$f_j(x_j) = \frac{R_j}{|I_j(x_j)|}\sum_{i\leq j}\int_{I_j(x_j)} s_i^*. \tag{9}$$

As in the proof of Lemma 5.3, we view $f_j(x_j)$ as the product of R_j with $\bar{s}^*(I_j)$ (the average value of s^* over interval $I_j(x_j)$). Since s^* is constant over any execution interval, $\bar{s}^*(I_j)$ attains its maximum when x_j is at a boundary point of some execution interval. □

Lemma 5.5 (Nesting) *Among all instances consistent with $(\mathcal{S},\gamma)$, the function F_A is maximized when the intervals I_j, $j \in J_A$, are properly nested: $I_i \cap I_j \neq \emptyset$ and $i < j$ implies $I_i \subseteq I_j$.*

Proof. Assume $I_i \cap I_j \neq \emptyset$ and $i < j$, but I_i and I_j are not nested. By the proof of Lemma 5.2, we must have $a_i > a_j$ we can write $I_j = pq$ and $I_i = qr$ for suitable subintervals p, q and r. The interpretation associated with $f_j(x_j)$ in Eq. 9 implies $\bar{s}^*(q) \geq \bar{s}^*(p)$, for otherwise $I_j = p$ would give $f_j(x_j)$ an even larger value. Similarly, we must have $\bar{s}^*(r) \geq \bar{s}^*(q)$. But then $\bar{s}^*(pqr) \geq \bar{s}^*(pq) = f_j(x_j)$, hence we can maximize f_j by letting $I_j = pqr$ instead, resulting in $I_i \subseteq I_j$. We repeat the process for each pair of intersecting intervals. Since the process causes the number of distinct endpoints to decrease, it will eventually terminate. □

In summary, by combining the preceding lemmas with Eq. 5, 7, we have established the following reduction to canonical instances.

Lemma 5.6 (Reduction) *Let J be a job instance optimized by s^*. Then there exists a schedule $\mathcal{S} = (s^*, job)$ and a bipartition γ such that*

$$\mathrm{AVR}(J) \leq 2 \limsup_{J'} \{F_A(J') + F_B(J')\}, \qquad (10)$$

where the lim sup *is taken over all canonical instances J' consistent with $(\mathcal{S},\gamma)$.*

5.2 Competitive Ratio

Lemma 5.6 reduces the analysis of AVR to the maximization of F_A, F_B over all canonical instances consistent with any $(\mathcal{S},\gamma)$. We will show that the latter problem can be represented in terms of the eigenvalues of a matrix determined by $(\mathcal{S},\gamma)$. We then prove a uniform bound on the largest eigenvalue of all such matrices.

Let $J = J_A \cup J_B$ be a canonical instance with $|J_A| = n$. Define

$$\delta_{ij} = \begin{cases} 1 & \text{if } I_i^* \subseteq I_j, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the δ_{ij}'s specify the intervals of the jobs in J_A by decomposing each I_j over the disjoint intervals $\{I_1^*, \ldots, I_n^*\}$. Note that the nesting property implies the following inequality for the δ_{ij}'s:

$$\delta_{ij}\delta_{ik} \leq \delta_{ij}\delta_{jk} \quad \text{for } i \leq j \leq k. \qquad (11)$$

Lemma 5.7 *For any canonical instance $J = J_A \cup J_B$, we have $F_A(J) \leq 4\ \mathrm{OPT}_A(J)$ and $F_B(J) \leq 4\ \mathrm{OPT}_B(J)$.*

Proof. It suffices to prove the inequality for J_A; hence all jobs considered here are assumed to be in J_A. We first compress the t-axis by deleting all execution intervals for type-B jobs. (This can only shorten I_j for some $j \in J_A$, hence increase $F_A(J)$ by Eq. 6.) For simplicity, assume all execution intervals for J_A have unit length, i.e., $|I_i^*| = 1$. (This restriction can be easily removed with the use of suitable normalizing factors; see remark after Eq. 15.) We thus have $s_j^* = R_j$, and

$$\mathrm{OPT}_A(J) = \sum_j (s_j^*)^2 |I_j^*| = \sum_j R_j^2. \qquad (12)$$

Also, we can write

$$\begin{aligned} f_j &= \frac{R_j}{|I_j|} \sum_{i \leq j} \delta_{ij} s_i^* \\ &= \frac{R_j}{|I_j|} \sum_{i \leq j} \delta_{ij} R_i. \end{aligned} \qquad (13)$$

Define a matrix $M(J_A)$ (or M for short) by

$$M_{ij} = \begin{cases} \delta_{ij}/|I_j| & \text{if } i \leq j, \\ 0 & \text{if } i > j. \end{cases}$$

In general, we call a matrix M of the above form a *tree-induced* matrix if it is induced by a set of properly nested intervals $\{I_j\}$ with $|I_j - \bigcup_{i<j} I_i| = 1$. For example, the matrix

$$M = \begin{pmatrix} 1 & 0 & 1/3 & 1/4 \\ 0 & 1 & 1/3 & 1/4 \\ 0 & 0 & 1/3 & 1/4 \\ 0 & 0 & 0 & 1/4 \end{pmatrix}$$

is induced by an instance of four intervals with lengths $|I_1| = |I_2| = 1$, $|I_3| = 3$, $|I_4| = 4$, and nesting relations $I_i \subseteq I_3 \subseteq I_4$ for $i = 1, 2$. Note that M has all column sums $\sum_{i \leq j} M_{ij}$ equal to 1. Also, by dividing both sides of Eq. 11 by $|I_j||I_k|$, we obtain the same nesting relations for the M_{ij}'s as for the δ_{ij}'s:

$$M_{ij}M_{ik} \leq M_{ij}M_{jk} \quad \text{for } i \leq j \leq k. \qquad (14)$$

Let $\mathbf{r}$ denote the vector $(R_1, \ldots, R_n)$ where $n = |J_A|$. We have

$$\begin{aligned} \mathrm{OPT}_A(J) &= \|\mathbf{r}\|^2 \\ F_A(J) &= 2\sum_j f_j \\ &= 2\left(\mathbf{r}M\mathbf{r}^t\right) \end{aligned} \qquad (15)$$

by Eq. 12 and 13. (When $|I_i^*| \neq 1$, Eq. 15 is still valid if we just multiply the j-th column of M by $|I_j^*|$, and divide the j-th component of $\mathbf{r}$ by $|I_j^*|^{\frac{1}{2}}$.) Hence, the proof of Lemma 5.7 will be complete if we show that $\mathbf{r}M\mathbf{r}^t \leq 2||\mathbf{r}||^2$. This is a consequence of the following Lemma. □

Let M^* be the symmetrized form of M: that is, $M^* = (M + M^t)/2$. Note that $\mathbf{v}M\mathbf{v}^t = \mathbf{v}M^*\mathbf{v}^t$ for all $\mathbf{v}$.

Lemma 5.8 *If M is a tree-induced matrix, then the largest eigenvalue of M^* is at most 2.*

Proof. For any $\mathbf{v} = (v_1, \ldots, v_n)$, let $\mathbf{w} = M\mathbf{v}^t = (w_1, \ldots, w_n)$; thus $w_i = \sum_{j \geq i} M_{ij} v_j$. It suffices to show that

$$||\mathbf{w}||^2 \leq 2(\mathbf{v} \cdot \mathbf{w}). \quad (16)$$

Indeed, by Cauchy's inequality, $(\mathbf{v} \cdot \mathbf{w}) \leq ||\mathbf{v}||||\mathbf{w}||$. This together with Eq. 16 gives $(\mathbf{v} \cdot \mathbf{w}) = \mathbf{v}M\mathbf{v}^t \leq 2||\mathbf{v}||^2$, which is what we want. We now prove Eq. 16 by making use of the nesting property of the M_{ij}'s as given by Eq. 14.

$$\begin{aligned} ||\mathbf{w}||^2 &= \sum_i w_i^2 \\ &= \sum_i \sum_{\substack{j \geq i \\ k \geq i}} M_{ij} M_{ik} v_j v_k \\ &\leq 2 \sum_i \sum_{k \geq j \geq i} M_{ij} M_{ik} v_j v_k \\ &\leq 2 \sum_i \sum_{k \geq j \geq i} M_{ij} M_{jk} v_j v_k \\ &= 2 \sum_j v_j (\sum_{k \geq j} M_{jk} v_k)(\sum_{i \leq j} M_{ij}) \\ &= 2(\mathbf{v} \cdot \mathbf{w}). \quad (17) \end{aligned}$$

□

The worst case of Lemma 5.8 occurs when the tree-induced matrix is $M_{ij} = 1/j$ for $i \leq j$, in which case $\mathbf{v} = \langle v_i \rangle = \langle 1/\sqrt{i} \rangle$ is asymptotically an eigenvector with eigenvalue approaching 2. This case corresponds exactly to the job instance given in Example 2 with $e = 3/2$.

We now complete the proof of the following theorem.

Theorem 2. *For the power function $P(s) = s^2$, the Average Rate Heuristic has competitive ratio r where $4 \leq r \leq 8$.*

Proof. The low bound follows from Example 2. The upper bound is an immediate consequence of Lemma 5.6 and 5.7. □

6 Higher-Order Power Functions

The approach we used to analyze AVR for the power function $P(s) = s^2$ generalizes readily to the case $P(s) = s^p$ when $p \geq 2$. The details will be left to the complete paper. As an illustration of the techniques used, we prove below the analog of Lemma 5.8. It is obtained by first extending Eq. 17 to p norms, and then using Hölder's inequality (a generalization of Cauchy's inequality).

Lemma 6.1 *Let M be a tree-induced matrix, and $\mathbf{w} = M\mathbf{v}^t$. Then $||\mathbf{w}||_p \leq p||\mathbf{v}||_p$ for any integer $p \geq 2$.*

Proof. We first generalize Eq. 17, again making use of the nesting property of the M_{ij}'s and the fact $\sum_{i \leq j} M_{ij} = 1$.

$$\begin{aligned} (||\mathbf{w}||_p)^p &= \sum_i w_i^p \\ &= \sum_i (\sum_{j \geq i} M_{ij} v_j)^p \\ &\leq \sum_i p \left(\sum_{j \geq i} M_{ij} v_j (\sum_{k \geq j} M_{ik} v_k)^{p-1} \right) \\ &\leq \sum_i p \left(\sum_{j \geq i} M_{ij} v_j (\sum_{k \geq j} M_{jk} v_k)^{p-1} \right) \\ &= p \sum_j v_j (\sum_{k \geq j} M_{jk} v_k)^{p-1} (\sum_{i \leq j} M_{ij}) \\ &= p \sum_j v_j w_j^{p-1}. \quad (18) \end{aligned}$$

By Hölder's inequality for p norms (see [4]), $\mathbf{u_1} \cdot \mathbf{u_2} \leq ||\mathbf{u_1}||_{p_1} ||\mathbf{u_2}||_{p_2}$ if $1/p_1 + 1/p_2 = 1$. Letting $\mathbf{u_1} = \mathbf{v}$, $\mathbf{u_2} = \langle w_j^{p-1} \rangle$, $p_1 = p$ and $p_2 = p/(p-1)$, we have

$$\begin{aligned} \sum_j v_j w_j^{p-1} &\leq ||\mathbf{v}||_p (\sum_j (w_j^{p-1})^{\frac{p}{p-1}})^{\frac{p-1}{p}} \\ &= ||\mathbf{v}||_p (\sum_j (w_j^p))^{\frac{p-1}{p}} \\ &= ||\mathbf{v}||_p (||\mathbf{w}||_p)^{p-1}. \quad (19) \end{aligned}$$

Lemma 6.1 thus follows from Eq. 18 and 19. □

However, the penalty factor we pay for analyzing J_A and J_B separately becomes 2^{p-1} since $(f+g)^p \leq 2^{p-1}(f^p+g^p)$. (The worst case of the inequality occurs when $f = g$.)

Theorem 3. *Let $p \geq 2$ be any real number. For the power function $P(s) = s^p$, the Average Rate Heuristic has a competitive ratio r_p satisfying $p^p \leq r_p \leq 2^{p-1}p^p$.*

7 Conclusion

As portable computing devices proliferate, we expect that the energy usage of computer systems will become an increasingly important problem. In this paper we have introduced a simplified model of variable speed processors and analyzed how the scheduling of jobs affects the overall power consumption. We have shown that the Average Rate heuristic has a constant competitive ratio for power function $P(s) = s^p$. While our bound p^p for the monotone case is tight, the general bound (i.e., for the bitonic case) involves a multiplicative factor of 2^{p-1}. Based on simulation results, we conjecture that the true competitive ratio is p^p.

For the heuristic Optimal-Available mentioned in Section 4, simulations also suggest that it has a competitive ratio of 4 for the $p = 2$ case, although we have not yet been able to prove any constant bound. Other heuristics, including some that appear to have better competitive ratios, will be discussed in the full paper.

One can show, by considering a simple two-job case, that there is a lower bound of 10/9 on the competitive ratios of all one-line scheduling algorithms. This bound can be improved slightly by using more sophisticated adversary strategies, and by considering three jobs, etc. It would be interesting to find a systematic approach for proving stronger lower bounds for this problem.

Finally, simulations of randomly generated instances (and simple probablistic arguments) suggest that the number of critical intervals grows rather slowly with n. This and other average case phenomena have yet to be investigated more fully.

Acknowledgements

We would like to thank Anant Sahai for performing extensive simulations on the various heuristics, which led to valuable insights to this problem and, in particular, uncovered the example with competitive ratio 4 described in this paper. We would also like to thank Dan Greene and Nabil Kahale for helpful discussions.

References

[1] W. Athas, J. Koller, and L. Svenson. An energy-efficient CMOS line driver using adiabatic switching. *1994 IEEE Fourth Great Lakes Symposium on VLSI*. University of Washington, 1993.

[2] A. Chandrakasan, S. Sheng, and R. Broderson. Low-power CMOS digital design. *JSSC*. 27 (4) 473-484, 1992.

[3] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. preprint.

[4] G. H. Hardy, Y. E. Littlewood, and G. Pólya. *Inequalities*, 1934.

[5] M Horowitz. Self-clocked structures for low power systems. Computer Systems Laboratory, Stanford University, ARPA semi-annual report, December 1993.

[6] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *CACM* 20 (1), 46-61, 1973.

[7] M Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. *Proc. Symposium on Operating Systems Design and Implementation*, pp. 13-23, 1994.

[8] S. Younis and T. Knight. Practical implementation of charge recovering asymptotically zero power CMOS. *1993 Symposium on Integrated Systems*, University of Washington, 1993.