# Min-Energy Scheduling for Aligned Jobs in Accelerate Model*

Weiwei Wu[1], Minming Li[2], and Enhong Chen[3]

[1] Department of Computer Science, University of Science and Technology of China
Department of Computer Science, City University of Hong Kong
USTC-CityU Joint Research Institute
wweiwei2@cityu.edu.hk

[2] Department of Computer Science, City University of Hong Kong
minmli@cs.cityu.edu.hk

[3] Department of Computer Science, University of Science and Technology of China
cheneh@ustc.edu.cn

**Abstract.** Dynamic voltage scaling technique provides the capability for processors to adjust the speed and control the energy consumption. We study the pessimistic accelerate model where the acceleration rate of the processor speed is at most $K$ and jobs cannot be executed during the speed transition period. The objective is to find a min-energy (optimal) schedule that finishes every job within its deadline. The job set we study in this paper is aligned jobs where earlier released jobs have earlier deadlines. We start by investigating a special case where all jobs have common arrival time and design an $O(n^2)$ algorithm to compute the optimal schedule based on some nice properties of the optimal schedule. Then, we study the general aligned jobs and obtain an $O(n^2)$ algorithm to compute the optimal schedule by using the algorithm for the common arrival time case as a building block. Because our algorithm relies on the computation of the optimal schedule in the ideal model ($K = \infty$), in order to achieve $O(n^2)$ complexity, we improve the complexity of computing the optimal schedule in the ideal model for aligned jobs from the currently best known $O(n^2 \log n)$ to $O(n^2)$.

## 1 Introduction

Energy-efficiency has become the first-class design constraint besides the traditional time and space requirements. Portable devices (like laptops and PDAs) equipped with capacity limited batteries are popular in our daily life. Two facts make the energy problem more important. First, the battery capacity is increasing with a rate less than that of power consumption of the processors. Second,

* This work was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 117408], National Natural Science Foundation of China (grant no. 60775037) and the National High Technology Research and Development Program of China (863 Program) (grant no. 2009AA01Z123).

the accumulated heat due to energy consumption will reach a thermal wall and challenge the designers of electronic devices. It is found that, in the CMOS processors, the energy consumption can be saved by executing with a lower speed. Approximately, the speed is a cubic root of the power, which is famous as cube-root-rule. Dynamic voltage scaling (DVS) technique is widely adopted by modern processor manufactures, e.g., Intel, AMD, and IBM. It allows the processor to dynamically adjust its voltage/frequency to control the power consumption. The first theoretical study was initialed decades ago [14], where the authors make the standard generalization, a speed to power function $P(s) = s^\alpha$ ($\alpha \geq 1$). Usually, $\alpha$ is 2 or 3 according to the cube-root-rule of the processors. From then on, lots of studies are triggered in this field. It is usually formulated as a dual objective problem. That is, while conserving the energy, it also needs to satisfy some QoS metric. When all jobs are required to be completed before deadline, the metric is called *deadline feasibility.* There are also works trying to simultaneously minimize the response time of the jobs, namely, *flow.* A schedule consists of the *speed scaling policy* to determine what speed to run at time $t$ and the *job selection policy* to decide which job to run at that time.

If the processor can run at arbitrary speeds, then based on how fast the voltage can be changed, there are two different models.

**Ideal Model:** It is assumed that the voltage/speed of the processor can be changed to any other value without any extra/physical cost or delay. This model provides an ideal fundamental benchmark and has been widely studied.

**Accelerate Model:** It is assumed that the voltage/speed change has some delay. In practice, the processor's acceleration capacity is limited. For example, in the low power ARM microprocessor system (lpARM) [5], the clock frequency transition takes approximately $25\mu s$ (1350 cycles) from 10MHz to 100MHz. Within this scope, there are two variations. In the *optimistic model*, the processor can execute jobs during the speed transition time, while in the *pessimistic model*, the execution of jobs in the transition time is not allowed [15].

## 1.1 Related Works

In recent years, there are many works on the impact of DVS technology. It is not practical to survey all of them, thus we just review the most related papers.

For the ideal model, Yao el. al. first studied the energy-efficient job scheduling to achieve deadline feasibility in their seminal paper [14]. They proposed an $O(n^3)$ time algorithm YDS to compute the optimal off-line schedule. Later on, the running time is improved to $O(n^2 \log n)$ in [12]. Another metric, the response time/flow, was examined in [13] with bounded energy consumption. It is first formulated as a linear single objective (energy+flow) optimization problem in [1]. This was then specifically studied in [4],[9],[6],[2],[3] el. al. under different assumptions. A good survey can be found in [8].

For the accelerate model, there are little theoretical studies to the best of our knowledge, except that the single task problem was studied in [7],[15]. In [7], they showed that the speed function which minimizes the energy is of some restricted shapes even when considering a single task. They also gave some empirical studies based on several real-life applications. In [15], the authors studied both the optimistic model and pessimistic model, but still for the single task problem. They showed that to reduce the energy, the speed function should accelerate as fast as possible from.

### 1.2 Main Contributions

In this paper, we study the pessimistic accelerate model to minimize the energy consumption. The QoS metric is deadline feasibility. The input is an aligned job set $\mathcal{J}$ with $n$ jobs, where jobs with earlier arrival times have earlier deadlines. The processor can execute a job with arbitrary speed but the absolute acceleration rate is at most $K$, and the processor has no capability to execute jobs during the transition of voltage. The objective is to find a min-energy schedule that finishes all jobs before their deadlines.

We first consider a special case of aligned jobs where all the jobs arrive at time 0. We call this kind of job set *common arrival time instance*. We prove that the optimal schedule should accelerate as fast as possible and the speed curve is non-increasing. Combining with other properties we observed, we construct an $O(n^2)$ time algorithm to compute the optimal schedule.

Then we turn to the general aligned jobs to study the optimal schedule $OPT_K$. The algorithm for the common arrival time instance is adopted as an elementary procedure to compute $OPT_K$. Most of the properties for the common arrival time instance can be extended to general aligned jobs. By comparing $OPT_K$ with the optimal schedule $OPT_\infty$ in the ideal model, we first prove that the speed curves of $OPT_K$ and $OPT_\infty$ match during some "peak"s. Then we show that the speed curve of $OPT_K$ between adjacent "peak"s can be computed directly. The whole computation takes $O(n^2)$ time since we improve the computation of $OPT_\infty$ for aligned jobs to $O(n^2)$. Our work makes a further step in the theoretical study of accelerate model and may shed some light on solving the problem for the general job set.

The organization of this paper is as follows. We review the ideal model and the pessimistic accelerate model in Section 2. In Section 3, we study the pessimistic accelerate model and focus on a special but significant case where all jobs are released at the beginning. We then turn to the general aligned jobs that have arbitrary release time in Section 4. Finally we conclude the paper in Section 5.

## 2 Model and Notation

In this section, we review the ideal model proposed in [14] and the pessimistic accelerate model.

The input job instance we consider in this paper is an aligned job set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ where each job $J_i$ has an arrival time $r(J_i)$, a deadline $d(J_i)$ (abbreviated as $r_i$ and $d_i$ respectively), and the amount of workload $C(J_i)$. The arrival times and the deadlines follow the same order, i.e., $r_1 \leq r_2 \leq \ldots \leq r_n$ and $d_1 \leq d_2 \leq \ldots \leq d_n$.

In the ideal model, the processor can change its speed to any value instantaneously without any delay. The power function is assumed to be $P(s) = s^{\alpha}(\alpha \geq 1)$. A schedule $S$ needs to determine what speed and which job to execute at time $t$. We use $s(t, S)$ to denote the speed took by schedule $S$ at time $t$ and write it as $s(t)$ for short if the context is clear. We use $job(t)$ to represent the index of the job being executed at time $t$. Jobs are preemptive. The processor has the capability to resume the formerly suspended jobs. We take the deadline feasibility as the QoS metric, i.e., a job is available after its arrival time and need to be completed before its deadline. A feasible schedule must satisfy the timing constraint $\int_{r_i}^{d_i} s(t)\delta(i, job(t))dt = C(J_i)$, where $\delta(i,j) = 1$ if $i = j$ and $\delta(i,j) = 0$ otherwise. The energy consumption is the power integrated over time: $E(S) = \int_t P(s(t,S))dt$. The objective is to minimize the total energy consumption while satisfying the deadline feasibility.

In the pessimistic accelerate model, the processor cannot change the voltage instantaneously. The acceleration rate is at most $K$, i.e., $|s'(t)| \leq K$. Moreover, no job can be executed during the transition interval $s'(t) \neq 0$ and there is always some job being executed when $s'(t) = 0$ and $s(t) > 0$. The energy is the power integrated over the time where $s'(t) = 0$ and $s(t) > 0$. So $E = \int_{t|s'(t)=0, s(t)>0} P(s(t,S))dt$. With such constraints, a *feasible* schedule is a schedule where all jobs are completed before deadline and the speed function satisfies $|s'(t)| \leq K$. The *optimal* schedule is the one with the minimum energy consumption among all feasible schedules.

Let $t_s = \min_i r_i, t_f = \max_i d_i$. The workload executed in interval $[a,b]$ by schedule $S$ is denoted as $C_{[a,b]}(S)$. If a job $J$ has $I(J) = [r(J), d(J)] \subseteq [a,b]$, we say $J$ is *embedded* in interval $[a,b]$. For simplicity, when we say "the first" time (or interval), we mean the earliest time (or interval) on the time axis in left-to-right order. Using the similar definition as [11], we say $t_u$ is a *tight deadline* (or *tight arrival time* respectively) in schedule $S$ if $t_u$ is the deadline (or arrival time respectively) of the job that is executed at $[t_u - \Delta t, t_u]$ (or $[t_u, t_u + \Delta t]$ respectively) in $S$ where $\Delta t \to 0$. Due to space limit, we omit most of the proofs in this version.

## 3 Optimal Schedules for Job Set with Common Arrival Time

For the jobs that have common arrival time, we assume w.l.o.g they are available at the beginning, namely $r_i = 0$ for $1 \leq i \leq n$.

**Definition 1.** *In a feasible schedule $S$, we denote the maximal interval where the jobs run at the same speed as a block. Note that there is an* acceleration-interval *(the time used for acceleration) between adjacent blocks because changing the speeds needs some time, during which no workload is executed.*

In the following, we will give some properties of the optimal schedule which help us design a polynomial algorithm to compute the optimal schedule.

**Lemma 1.** *There exists an optimal schedule, where the speed function is non-increasing and will accelerate as fast as possible, i.e., either* $|s'(t)| = K$ *or* $|s'(t)| = 0$*; and the jobs are completed in EDF (Earliest Deadline First) order with* $J_i$ *executed in one speed and only in the lowest speed* $\min\limits_{0 \le t \le d_i, s'(t)=0} s(t)$*. Furthermore, the finishing time* $\hat{t}$ *of each block (where* $\lim_{t \to \hat{t}^-} s'(t) = 0 \wedge \lim_{t \to \hat{t}^+} s'(t) = -K$*) is a tight deadline.*

**Lemma 2.** *In the optimal schedule,*

*1) The first block is the interval* $(0, d_t)$ *which maximizes* $\frac{\sum_{J \in \mathcal{J}_t} C(J)}{d_t}$ *where* $\mathcal{J}_t = \{J_j | d_j \le d_t]\}$ *and* $t \in \{1, \dots, n\}$*, i.e. the maximum speed in the optimal schedule is* $s_1 = \max\limits_i \frac{\sum_{J \in \mathcal{J}_i} C(J)}{d_i}$.

*2) Suppose block* $j$ *has speed* $s_j$ *and finishes at* $J_{t_j}$*'s deadline, then the speed in block* $j+1$ *is* $s_{j+1} = \max\limits_t \frac{s_j - K(d_t - d_{t_j}) + \sqrt{(K(d_t - d_{t_j}) - s_j)^2 + 4K \sum_{i=t_j+1}^{t} C(J_i)}}{2}$ *where* $t \in \{t_j + 1, \dots, n\}$.

**Theorem 1.** *The optimal schedule can be computed by Algorithm 1 at* $O(n^2)$*.*

*Proof.* Algorithm 1 is a direct implementation of Lemma 2. Steps 2-4 computes the first block. The two loops in Steps 6-10 computes the remaining blocks. By keeping the information of the summation on the computed jobs, the optimal schedule can be computed in $O(n^2)$ time.

**Algorithm 1.** *CRT_schedule*

```
1. $t = 0$;
2. $s_1 = \max\limits_i \frac{\sum_{j=1}^{i} C(J_j)}{d_i}$;
3. $t = \arg\max\limits_i s_1$ ;
4. Let the block with speed $s_1$ be $[0, d_t]$;
5. $m = 1$;
while $t < n$ do
    6. $s_{m+1} = \max\limits_{t+1 \le i \le n} \frac{s_m - K(d_i - d_t) + \sqrt{(K(d_i - d_t) - s_m)^2 + 4K \sum_{j=t+1}^{i} C(J_j)}}{2}$;
    7. $t' = \arg\max\limits_i s_{m+1}$;
    8. Let the block with speed $s_{m+1}$ be $[d_t + (s_m - s_{m+1})/K, d_{t'}]$;
    9. $m = m + 1$;
    10. $t = t'$;
end while
```

## 4 Optimal Schedules for Aligned Jobs

In this section, we study the optimal schedule for general aligned jobs. Note that jobs with common arrival time is a special case of aligned jobs. We first extend some basic properties in Subsection 4.1. We will compute the optimal schedule for aligned jobs by adopting Algorithm 1 as a building block. We use $OPT_K$ to denote the optimal schedule where $K$ is the maximum acceleration rate.

In the ideal model, the acceleration rate is infinity $K = \infty$. We review the Algorithm YDS in [14] to compute $OPT_\infty$. Let $w(t_1, t_2)$ denote the workload of the jobs that have release time at least $t_1$ and have deadline at most $t_2$, i.e. $w(t_1, t_2) = \sum_{I(J)\subseteq[t_1,t_2]} C(J)$. Define the intensity $Itt(t_1, t_2)$ of the time interval $[t_1, t_2]$ to be $w(t_1, t_2)/(t_2 - t_1)$. The algorithm tries every possible pair of arrival time and deadline to find an interval with largest intensity (called *critical interval*), schedule the jobs embedded in the critical interval and then repeatedly deal with the remaining jobs.

We first show that the optimal schedule for aligned jobs in the ideal model can be computed in $O(n^2)$ time.

**Theorem 2.** *The optimal schedule for aligned jobs in the ideal model can be computed in $O(n^2)$ time.*

Given a block $block_p$, we denote the corresponding interval as $[L(block_p), R(block_p)]$. We define *virtual canyon* to be a block with length 0. Next, we derive some properties of $OPT_K$.

## 4.1 Basic Properties

Among all the blocks, we define the block $[t_a, t_b]$ where $\lim_{t\to t_a{}^-} s'(t) = K \wedge \lim_{t\to t_a{}^+} s'(t) = 0$ and $\lim_{t\to t_b{}^-} s'(t) = 0 \wedge \lim_{t\to t_b{}^+} s'(t) = -K$ to be *peak*. Reversely, the block where $\lim_{t\to t_a{}^-} s'(t) = -K \wedge \lim_{t\to t_a{}^+} s'(t) = 0$ and $\lim_{t\to t_b{}^-} s'(t) = 0 \wedge \lim_{t\to t_b{}^+} s'(t) = K$ is called *canyon*.

We say $\hat{t}$ is *down-edge-time* if $\lim_{t\to \hat{t}^-} s'(t) = 0 \wedge \lim_{t\to \hat{t}^+} s'(t) = -K$ or $\lim_{t\to \hat{t}^-} s'(t) = K \wedge \lim_{t\to \hat{t}^+} s'(t) = 0$. For example, both the start time and finish time of a peak are down-edge-times.

We have the following lemma for $OPT_K$.

**Lemma 3.** *There is an optimal schedule, where the speed function will accelerate as fast as possible, i.e., either $|s'(t)| = K$ or $s'(t) = 0$, and every down-edge-time is either a tight deadline or a tight arrival time; and jobs are executed in EDF order; each job $J$ is executed only in one block, and this block is the lowest one in interval $[r(J), d(J)]$.*

## 4.2 $O(n^2)$ Time Algorithm to Compute $OPT_K$

To find the optimal schedule, our method is to identify some special blocks belonging to $OPT_K$. After enough blocks are selected, the remaining interval of $OPT_K$ can be easily computed. To be more specific, we compare $OPT_K$ with schedule $OPT_\infty$, which is the optimal schedule for the special case $K = \infty$, namely the ideal model. We observe that the block with the highest speed (we call it *global-peak*) of $OPT_K$ can be computed first.

**Algorithm 2.** Computing a Monotone-interval

```
Input: $OPT_\infty$, Schedule computed by YDS.
$[a,b]$, computed peak (it can be the global-peak or local-peak)
$s_b$, Starting speed at time $b$.
Output: $S_{[b,t_1]}$, a monotone-interval starting from $b$ and its corresponding schedule.
/* Let $[t_L, t_R]$ be the current interval being handled. Let $S$ be the the
computed schedule for current interval $[t_L, t_R]$. $s_{last}$ denotes the lowest
speed in the computed $S$. $block_{p+1}$ is the first un-handled block in $OPT_\infty$.*/
1. $s_{last} = s_b$; $t_L = t_R = b$; $S_{[b,t_1]} = \phi$; $S = \phi$; $p = 0$; $\overline{t} = b$.
2. In $OPT_\infty$, index the blocks from the peak $[a,b]$ as $block_0, block_1, block_2, \dots$ in the
left to right order.
while $s_{last} \geq s(block_{p+1})$ do
  /*recover procedure*/
  if $S \neq \emptyset$ then
    3. For jobs that are executed in the lowest block of $S$, recover their arrival
    time/deadline to the original value.
    4. Reset $s_{last}$ to be the speed of $S$ in time $\overline{t}$;
  end if
  5. Select $block_i$ to be the block after $t_R$ in $OPT_\infty$ with i.e. $s(block_{p+1}) > \dots >$
  $s(block_i)$ and $s(block_i) < s(block_{i+1})$; if such a block does not exist, then let
  $i = p + 1$; Reset $t_R = R(block_i)$.
  6. Set $p = i$;
  /*adjust procedure*/
  for every job with $I(J) \cap [t_L, t_R] \neq \phi$ do
    7. Adjust $r(J)$ to be $\max\{r(J), t_L\}$;
    8. Adjust $d(J)$ to be $\min\{d(J), t_R\}$;
    9. Backup the original value of $r(J)$ and $d(J)$;
  end for
  /*handle interval $[t_L, t_R]$ in $OPT_\infty$*/
  10. Call Algorithm 1 to compute a schedule $S$ for jobs involved in Step 9 ac-
  cording to common arrival time $t_L$ with starting speed $s_{last}$.
  11. If the $block_i$ found in Step 6 has speed 0, then we make $S$ accelerate with rate
  $-K$ after the last time with positive speed and insert a virtual canyon at time $t_R$.
  12. Reset $s_{last}$ to be the lowest positive speed in the computed $S$.
  if $s_{last} < s(block_{p+1})$ then
    13. $S_{[b,t_1]} = S_{[b,t_1]} \cup S$; Return $S_{[b,t_1]}$.
  else
    14. Let $\overline{t}$ be the finish time of the second lowest (including the virtual canyon
    inserted in Step 11) block in $S$.
    15. $S_{[b,t_1]} = S_{[b,t_1]} \cup$ ($S$ restricted in interval $[t_L, \overline{t}]$).
    16. Reset $t_L = \overline{t}$.
  end if
end while
```

**Lemma 4.** *$OPT_K$ executes the same as $OPT_\infty$ in the first critical interval.*

After we have fixed the first block (global-peak) of $OPT_K$, a natural question is whether we can apply the same proof of Lemma 4 to select other blocks. For example, in the remaining interval of $OPT_\infty$, does the block with maximum

**Algorithm 3.** Computing the Optimal Schedule between Two Adjacent Peaks

```
**Input:**
$[a_1, b_1], [a_2, b_2]$, the two adjacent peaks found in Algorithm 4.
$S_{[b_1,t_1]}, S_{[t_2,a_2]}$, the two schedules computed by Algorithm 2. Choose one of the intersection point as $\bar{t}$.
**Output:** schedule of $OPT_K$ in interval $[b_1, a_2]$.
1. For each down-edge-time $p$ on $s(t, S_{[b_1,t_1]})$ in $[b_1, \bar{t})$ or on $s(t, S_{[t_2,a_2]})$ in $(\bar{t}, a_2]$, let the point with the same speed on the other curve be $p'$. If there are more than one such point, let $p'$ be the one minimizing $|pp'|$; if there are no such point, we do not consider line segment originating from $p$.
2. Sort all the segments $pp'$ by increasing order of their speed (denoted by $Speed(p)$) into $p_1p'_1, p_2p'_2, \ldots, p_mp'_m$ (Duplicate segments are treated as one). The end points are relabeled so that $p_i$ is always on $s(t, S_{[b_1,t_1]})$ and $p'_i$ is always on $s(t, S_{[t_2,a_2]})$.
3. Find augment segment for each segment $p_ip'_i$ as follows. If $p_i$ and $p'_i$ are both down-edge-time, then the augment segment is $p_ip'_i$ itself; if $p_i$ is a down-edge-time and $p'_i$ is not, then the augment segment is $p_ip'$ where $p'$ is the closest down-edge-time on $s(t, S_{[t_2,a_2]})$ with respect to $p'_i$; the remaining case is similarly defined. We use $q_iq'_i$ to represent the augment segment of $p_ip'_i$.
**for** $i = 1$ to $m$ **do**
    4. Let $C = \sum_{I(J)\cap[q_i,q'_i]\neq\emptyset} C(J)$.
    **if** $(\frac{C}{|p_ip'_i|} < Speed(p_i))$ **then**
        5. Let $S_{[\hat{t}_1,\hat{t}_2]}$ be the schedule that executes all jobs with $I(J) \cap [p_i, p'_i] \neq \phi$ with speed $s$ in interval $[\hat{t}_1, \hat{t}_2]$.(The parameters can be calculated as $\hat{t}_1 = p_i+T$; $\hat{t}_2 = p'_i - T$; $s = Speed(p_i) - 2KT$; $T = \frac{Speed(p_i)+K|p_ip'_i|-\sqrt{(Speed(p_i)-K|p_ip'_i|)^2+4KC}}{4K}$)
        6. **break**;
    **end if**
**end for**
7. The optimal schedule in interval $[b_1, a_2]$ is ($S_{[b_1,\hat{t}_1]}$ restricted to $[b_1, \hat{t}_1]$) $\cup\, S_{[\hat{t}_1,\hat{t}_2]} \cup$ ($S_{[t_2,a_2]}$ restricted to $[\hat{t}_2, a_2]$).
```

intensity have the same schedule as that of $OPT_K$? Although this is not true, we will show that some other blocks in $OPT_\infty$ can be proved to be the same as $OPT_K$. The key observation is that by appropriately dividing the whole interval into two sub-intervals, the block with the maximum intensity inside one of the sub-intervals in $OPT_\infty$ can be proved to be the same as $OPT_K$. Our partition of intervals is based on a *monotone-interval* defined below.

**Definition 2.** *Given a schedule, we define the sub-interval where the speed function/curve is strictly non-increasing or non-decreasing to be a monotone-interval.*

Since the speed in $OPT_K$ outside the global-peak $[a, b]$ is at most $Itt(a, b)$, there exists a monotone-interval immediately after time $b$ (non-increasing curve) and symmetrically before time $a$ (non-decreasing curve). At time $b$ and $a$, the speeds are respectively $s_b = Itt(a, b)$ and $s_a = Itt(a, b)$.

In the following, we will study a schedule $S_{[b,t_1]}$ (only specifying speeds in interval $[b, t_1]$) with monotone-interval $[b, t_1]$ (non-increasing speed with

**Algorithm 4.** Computing the Optimal Schedule for Aligned Jobs

```
Input: Aligned job set $\mathcal{J}$
Output: $OPT_K$
1. Compute $OPT_\infty$.
2. Let the maximum intensity block in $OPT_\infty$ be the global-peak in $OPT_K$ .
3. Index the global-peak as an un-handled peak.
while there is a peak [L,R] un-handled do
    4. Let $OPT_K$ execute jobs the same way as $OPT_\infty$ in $[L, R]$.
    5. Call Algorithm 2 to compute the monotone-interval starting from $R$ (and also
    symmetrically a monotone-interval ending at $L$).
    6. If there are local-peaks in $OPT_\infty$ in the un-handled interval on either side of
    the monotone-intervals, then index the local-peaks as un-handled peaks.
end while
7. Compute the $OPT_K$ for all the intervals between adjacent peaks found in the
previous while loop using Algorithm 3.
```

$s(b, S_{[b,t_1]}) = s(b, OPT_\infty))$. Suppose that $t_1$ is the first (earliest) intersection of the two curves $s(t, S_{[b,t_1]})$ and $s(t, OPT_\infty)$ with $\lim_{t\to t_1^+} s(t, OPT_\infty) > 0$. We will compare the speed curve of $OPT_K$ with that of $S_{[b,t_1]}$.

**Definition 3.** *In interval $[b, t_1]$, we say $t$ is a separation-time of $OPT_K$ w.r.t $S_{[b,t_1]}$ if their speed curves totally overlap in interval $[b, t]$ and separate at $t + \Delta t$ where $\Delta t \to 0$.*

We can show that the schedule $S_{[b,t_1]}$ with non-increasing speed output by Algorithm 2 has the following property: let $[a_2, b_2]$ be the maximum intensity block in $OPT_\infty$ among the remaining interval $[t_1, t_f]$, then $OPT_K$ has the same schedule as $OPT_\infty$ in interval $[a_2, b_2]$. Furthermore, Algorithm 2 runs in $O(n^2)$.

Among the un-handled interval (e.g. $[t_1, t_f]$), we define *local-peak* to be the peak which has the local maximal intensity in $OPT_\infty$. The following lemma shows that the schedules $OPT_K$ and $OPT_\infty$ are the same in local-peaks.

**Lemma 5.** *The schedule of local-peaks in $OPT_K$ is the same as $OPT_\infty$.*

Note that there is a monotone-interval respectively before and after the computed global-peak or local-peak. We can repeatedly call Algorithm 2 (a symmetric version of Algorithm 2 can be used to compute a monotone-interval before a "peak") until no such peak exists in the un-handled intervals. Then the schedule of the remaining intervals (all intervals between the adjacent peaks computed in Algorithm 4) can be uniquely computed as shown in Lemma 6.

**Lemma 6.** *The schedule of $OPT_K$ in intervals between two (local-)peaks found by Algorithm 4 can be computed by Algorithm 3. Notice that in this algorithm, "down-edge-time" means the corresponding point on the speed curve at the down-edge-time.*

**Theorem 3.** *Algorithm 4 computes $OPT_K$ for aligned jobs in $O(n^2)$ time.*

## 5 Conclusion

In this paper, we study the energy-efficient dynamic voltage scaling problem and mainly focus on the pessimistic accelerate model and aligned jobs. All jobs are required to be completed before deadlines and the objective is to minimize the energy. We start by examining the properties for the special case where jobs are released at the same time. We show that the optimal schedule can be computed in $O(n^2)$. Based on this result, we study the general aligned jobs. The algorithm for jobs with common arrival time is adopted as an elementary procedure to compute the optimal schedule for general aligned jobs. By repeatedly computing heuristic schedules that is non-increasing, we fix some peaks of the optimal schedule first. This makes the optimal schedule in the remaining interval easier to compute. The complexity of the algorithm is $O(n^2)$ since we improve the computation of the optimal schedule for aligned jobs in the ideal model to $O(n^2)$.

## References

1. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 621–633. Springer, Heidelberg (2006)
2. Bansal, N., Chan, H.L., Lam, T.W., Lee, L.K.: Scheduling for bounded speed processors. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 409–420. Springer, Heidelberg (2008)
3. Bansal, N., Chan, H.-L., Pruhs, K.: Speed scaling with an arbitrary power function. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (SODA), pp. 693–701 (2009)
4. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (SODA), pp. 805–813 (2007)
5. Burd, T.D., Pering, T.A., Stratakos, A.J., Brodersen, R.W.: A dynamic voltage scaled microprocessor system. In: IEEE International Solid-State Circuits Conference, February 2000, pp. 294–295, 466 (2000)
6. Chan, H.-L., Edmonds, J., Lam, T.-W., Lee, L.-K., Marchetti-Spaccamela, A., Pruhs, K.: Nonclairvoyant speed scaling for fow and energy. In: STACS, Freiburg, Germany, pp. 409–420 (2009)
7. Hong, I., Qu, G., Potkonjak, M., Srivastava, M.B.: Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In: Proceedings of the IEEE Real-Time Systems Symposium (RTSS), pp. 178–187 (1998)
8. Irani, S., Pruhs, K.R.: Algorithmic problems in power management. SIGACT News 36(2), 63–76 (2005)
9. Lam, T.W., Lee, L.-K., To, I.K.-K., Wong, P.W.H.: Speed scaling functions for fow time scheduling based on active job count. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 647–659. Springer, Heidelberg (2008)
10. Li, M., Liu, B.J., Yao, F.F.: Min-energy voltage allocation for tree-structured tasks. Journal of Combinatorial Optimization 11(3), 305–319 (2006)
11. Li, M., Yao, F.F.: An efficient algorithm for computing optimal discrete voltage schedules. SIAM J. on Computing 35, 658–671 (2005)

12. Li, M., Yao, A.C., Yao, F.F.: Discrete and continuous min-energy schedules for variable voltage processors. Proc. of the National Academy of Sciences USA (PNAS) 103, 3983–3987 (2006)
13. Pruhs, K., Uthaisombut, P., Woeginger, G.: Getting the best response for your erg. In: Scandanavian Workshop on Algorithms and Theory, pp. 14–25 (2004)
14. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. IEEE Symp. Foundations of Computer Science (FOCS), pp. 374–382 (1995)
15. Yuan, L., Qu, G.: Analysis of energy reduction on dynamic voltage scaling-enabled systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24(12), 1827–1837 (2005)