Abstract—Reducing power or energy consumption to extend battery lifetime of portable systems, decrease chip cooling costs, and increase system reliability become one of the major concerns in today's embedded system design. Maximizing battery lifetime is difficult due to nonlinearities of charge delivery, especially when the load varies with time. Dynamic voltage scaling (DVS) is a promising technique for battery-powered system to conserve energy consumption. Since DVS scheduling problem where the target processor operates at discrete voltage is well known to be NP-hard in general, we propose a heuristics algorithm and a genetic algorithm (GA) which are suitable to solve such problem. To evaluate the effectiveness of the approaches, we choose synthetic examples of periodic and aperiodic tasks which are excerpted from the comparative work, on uniprocessor or multiprocessor platforms. Our experimental results show that the proposed scheduling algorithms significantly reduces up to 19% of dynamic energy consumption compared with a past approach. In addition, we also conduct experiments with randomly generated tasks and practical application. The experiment results of randomized experiments and practical application experiments show the superiority of the proposed algorithms not only in charge saving but also in schedulability ratio.

I. INTRODUCTION

Low energy consumption is a key design requirement in the battery-powered portable devices such as mobile computing and wireless communication applications in recent years. The problem is quite challenging due to the nonlinear behavior of the battery, since the amount of energy delivered by the battery depends on the discharge current profile [1]. Reducing power or energy consumption can extend battery lifetime of portable systems, decrease chip cooling costs, and increase system reliability [2]. DVS is a promising technique for battery-powered system to conserve energy consumption. In this paper, we proposed two algorithms base on DVS processors to obtain an optimum discharge current profile whose residual charge or the battery voltage is maximized on both uniprocessor and multiprocessor DVS systems.

DVS-based real-time scheduling for energy savings can be classified into static scheduling algorithms and dynamic scheduling algorithms. For static scheduling algorithms, all the information of the tasks such as periods, executive times and deadlines are known before the execution of the tasks. For dynamic scheduling algorithms, the parameters of all the tasks are not known until the execution time. Strategies that have been developed to reduce the energy consumption are affected by battery behavior. A pulsed discharge allows charge recovery during the idle periods. Recovery depends on the state of charge of the battery and on the duration of the rest time period.

In recent years, there has been significant amount of work done in studying battery behavior. There are two kinds of level analysis of battery behavior. A detailed low-level analysis of the main physical processes taking place inside a complex electrochemical system is important to be known for a battery designer. However, for a battery user it is important to know the battery behavior at the macroscopic scale. Therefore, a high-level model should capture only the gross characteristics of the battery behavior. Although computationally expensive low-level models (e.g., Dualfoil [3]) are accurate, high-level models provide reasonably accurate approximation in lightweight computation [1],[4]. The accuracy of a high-level analytical battery model developed by Rakhmatov and Vrudhula is confirmed by the low-level electrochemical simulation Dualfoil within approximately 5% error according to their report.

The battery characteristics have been widely studied [1],[4],[5] to shape the discharge profile [6]-[9]. Rakhmatov and Vrudhula specified various important properties of analytical model [4], and proposed efficient static battery-aware voltage scheduling algorithms combining of recovery insertion and voltage scaling [7]. Chowdhury and chakrabarti [1] developed several key properties of the cost function with respect to voltage scaling in task scheduling and extended the work of Rakhmatov, Vrudhula and Chakrabarti’s [7]. An heuristic algorithm they had proposed has been applied to both single-processor and multiprocessor systems. The algorithm first creates a task sequence by earliest deadline first (EDF) algorithm to ensure battery survival, then distributes the available delay slack so that the residual charge is maximized. EDF algorithm is also proved that the energy consumption of multiprocessor is minimize in case of the workload is perfectly balanced among the processor cores [10],[11]. Yokoyama and Takada [12] proposed heuristics algorithms based on several battery properties and Lagrange multipliers, the algorithms showed superior results on synthetic examples of periodic and aperiodic tasks from the task sets which are excerpted from the comparative work.

A charge-based cost function derived from analytical im-
important battery properties has been applied in their works [1,7,12]. In this paper, we have proved some properties of battery which can be applied to task-scheduling and improved the cost function with respect to voltage scaling in task scheduling. Since DVS scheduling problem where the target processor operates at discrete voltage is well known to be \( NP \)-hard in general, we have improved the heuristics algorithm in [12] and proposed a genetic algorithm (GA) which is suitable to solve the \( NP \)-hard problem. We analyzed the complexity of the proposed genetic algorithms. To evaluate the effectiveness of the approaches, we have chosen synthetic examples of periodic and aperiodic tasks which had been excerpted from the comparative work or had been generated randomly, on uniprocessor or multiprocessor platforms. Our experimental results show that the proposed scheduling algorithms significantly reduced up to 19% of dynamic energy consumption compared with a past approach.

The remainder of this paper is organized as follows: In Sect. 2 some motivation examples are given to reveal the factors related to energy minimizing. Sect. 3 describes some related work (e.g., Battery model, System configuration). Sect. 4 discusses in detail of our approaches. In Sect. 5, some comparative experimental tests are carried out and some concluding remarks follow in Sect. 6.

II. MOTIVATION

A. Illustrative Examples

In order to motivate the need for battery-aware system design, an example is presented in Fig. 1. There are four identical tasks (current 300 mA, duration 2 min in the voltage 3.3 V and deadline 30 min) are used to reveal the discharge characteristics of the battery on a battery-powered uniprocessor system. The processor speed and power change continuously. The consumed capacity is measured by the objective function \( \sigma_T \) (see Sect. 3 in detail), higher \( \sigma_T \) denotes the smaller residual charge at observation time \( T \).

It is well known that the time between the end of task execution and deadline is called slack time. The objective function \( \sigma_T \) can be minimized by using slack time. Due to non-linearity of the objective function (see section III in detail), using slack time not always improve the residual charge well. Fig.1(b) demonstrates that non-decreasing profile does minimized the residual charge. However, Fig.1(d) is an optimal profile with the minimum \( \sigma_{30} \) (a 25.2% improvement to the level current profile and a 27.3% improvement to the increase current profile). This implies voltage scaling can be used to reduce power consumption, but it is not always performance well.

Especially, non-decreasing and non-increasing current profile are shown in Fig.1(b) and Fig.1(c) which are obtained by scaling down the same voltage and the same current for the same task set (in Fig.1(a)). From Fig.1, the non-increasing profile does significantly better compared to non-decreasing current profile for the task set duration of only 30 min. This implies that based on modifying the discharge current profile can improve the battery performance.

\[
\frac{\Delta^*}{\Delta} = s \left( 1 + \frac{2(s - 1)V_{th}}{V_{dd} - V_{th}} \right)
\]

where \( \Delta^* \) denotes the new task duration. \( V_{th} \) denotes the threshold voltage whose value is set as 0.4V in this paper. The DVS processor can change the voltage and speed according to system requirements. Assume that the operating voltage ranges over set \{3.3, 3.0, 2.7, 2.5, 2.0\}, consistent with the voltage range of the Strong ARM SA1100 DVS processor. The battery current \( I_{batt} \) scales by \( s^3 \), i.e.,

\[
\frac{I_{batt}^*}{I_{batt}} = s^3
\]

where \( I_{batt}^* \) denotes the current after scaling the task voltage. For the sake of simplicity, static energy consumption is ignored for comparison purpose.
B. Battery Models

Charge and voltage are two important factors in battery models, yet no single model can handle both charge and voltage at the same time. In this paper we use the high-level analytical model presented by Rakhmatov and Vrudhula [4],[5] to focus on charge only. With the time-varying current $i(t)$ and lifetime $L$, the battery model is shown as follow:

$$\alpha = \int_0^L i(\tau)d\tau + 2\sum_{m=1}^{\infty} \int_0^L i(\tau)e^{-\beta^2m^2(L-\tau)}d\tau$$  \hspace{1cm} (3)

For a given battery, $\alpha$ (the theoretical capacity of the battery) and $\beta$ (the recovery rate of the battery) are constants. When current $i(t)$ does not vary with time, in other words, $i(t)$ changes into a constant ($i(t) = I$), the equation (3) is redeclared as:

$$\alpha = I[L + 2\sum_{m=1}^{\infty} \frac{1-e^{-\beta^2m^2L}}{\beta^2m^2}]$$ \hspace{1cm} (4)

The second term of (4) is nearly zero when $\beta$ become infinite. In this case, we obtain an ideal battery model with linear characteristic. Equation (4) brings the range set of $m$ in the second term to our attention. Some experiments have been carried on in [4],[5] to show that the approximation of the first 10 terms is sufficient for accurate lifetime predictions.

For comparison purpose, we use the same battery parameters in all profiles in this paper as in other references [1], [15] where $\alpha$ is 40 375mA-min and $\beta$ is 0.273 $\text{min}^{-1/2}$.

C. Cost Function

Assume that the profile consists of $n$ scheduled tasks. The $k$th task consists of the current $I_k$, the starting time $t_k$, and the duration $\Delta_k$. Understandable, $t_k + \Delta_k$ is the finish time of the $k$th task. The charge that lost at observation time $T$ is:

$$\sigma_T = \sum_{k \in \tau} I_k F(T, t_k, t_k + \Delta_k, \beta)$$ \hspace{1cm} (5)

in which $F$ is a function whose expression is:

$$F(T, s, f, \beta) = f - s + 2\sum_{m=1}^{m_{max}} \frac{e^{-\beta^2m^2(T-f)} - e^{-\beta^2m^2(T-s)}}{\beta^2m^2}$$ \hspace{1cm} (6)

where $s$ denotes the start time (i.e.,$t_k$) and $f$ denotes the finish time (i.e.,$t_k + \Delta_k$) of the single task. Since $m_{max}$ is infinite, the number of terms in the sum of infinite series provides a tradeoff between the accuracy and the amount of computation. As above description, the first 10 terms are a good approximation.

The profile quality metric defined as $Q = \alpha - \sigma$ is the cost function to be maximized. Note that a negative $Q$ at the end of a profile indicates battery failure.

D. Problem Description

In order to indicate our algorithms, a brief description of the problem is given as follows. Given the battery parameters $\alpha$ and $\beta$. An application consists of a set of task graphs $G = (\tau, E)$, where $\tau = \{\tau_1, \cdots, \tau_m\}$ is a finite set of task vertices (i.e., nodes), $E$ is a finite set of task dependencies representing connections between these nodes. Assume there are $M$ processors in the embedded system and no energy overhead by interprocessor communication. Each node $\tau_i$ is allocated to uniprocessor or multiprocessor system, and has a known worst case execution time $C_i$, a period $T_i$, a deadline $D_i$, frequency $\Omega_i$. In this paper, we do not consider task preemption and the length of profile must be smaller than observation time $T$. Assume that the operating voltage ranges over set $V = \{V_1, \cdots, V_g\}$, Output is a schedule or discharge profile consists of $n$ scheduled tasks such that the deadline and precedence constraints are met and profile quality metric $Q$ is maximized.

IV. Schedulability Analysis

The power consumption of circuit of CMOS can be widely divided into static power consumption and dynamic power consumption. The main power consumption is dynamic power consumption which can be illustrated as follow:

$$P_{\text{dynamic}} = \alpha \times C \times V_{dd}^2 \times f$$ \hspace{1cm} (7)

where $C$ is the load capacitor ($\alpha \times C$ denotes the switching capacitor), $V_{dd}$ is the working voltage and $f$ is the switching frequency. Since power is proportional to $V_{dd}^2$, clearly a slight reduction in the supply voltage can lead to a significant reduction in the power consumption [13]. Unfortunately reducing the supply voltage also increases the delays of the circuit, thus decreasing the achievable clock frequency. In static CMOS, power dissipation is dominated by switching power which is proportional to the square of the supply voltage [14]. The penalty of reduced voltage is increased propagation delays proportional to:

$$\Delta = \frac{k \times V_{dd}}{(V_{dd} - V_{th})^2}$$ \hspace{1cm} (8)

where $V_{th}$ is the transistor threshold voltage and $k$ is proportionality constant. It is well known that the reciprocal relationship between $f$ and $\Delta$ (i.e.,$f = 1/\Delta$). Assume that $V_{dd} \approx (V_{dd} - V_{th})$. Therefore (7) can be redeclared as:

$$P_{\text{dynamic}} = \frac{a \times C \times (V_{dd} - V_{th})^3}{k}$$ \hspace{1cm} (9)

Let $E$ denotes the energy consumption. And $\sigma$ can be described as:

$$E = P_{\text{dynamic}} \times \Delta = \frac{a \times C \times (V_{dd} - V_{th})^3}{k} \times \Delta$$ \hspace{1cm} (10)
We change the expression of $E$ as follow for simplify.

$$E = k \times V_{dd}^3 \times \Delta$$  \hspace{1cm} (11)

Since $\sigma = E/V_{dd}$, so another expression of $\sigma$ is got:

$$\sigma = k \times V_{dd}^2 \times \Delta$$  \hspace{1cm} (12)

It should be noted that (12) is the origin format of (5). (5) is the power consumption equation of specific battery model.

**Theorem 1**: [The most effective theorem of decreasing to the next lower level] Given a set of $n$ independent tasks, scaling the voltage of the tasks to the next level is more effective than to any other levels.

**Proof**: In (1), as $V_{dd} \in \{3.3, 3.0, 2.7, 2.5, 2.0\}$, we can easily conclude that the range of $s$ is from 1.08 to 1.65. Then the range of $1 + 2(s - 1)V_{th}/(V_{dd} - V_{th})$ is from 1.03 to 1.18. In this proof we regard $1 + 2(s - 1)V_{th}/(V_{dd} - V_{th})$ as constant 1 for simplify. In this condition (1) can be redeclared as:

$$\frac{\Delta^*}{\Delta} \approx s = \frac{v}{v^*}$$  \hspace{1cm} (13)

- Let $\Delta$ denotes the duration of the task before scaling the voltage of the task.
- Let $\Delta_1$ denotes the duration of the task after scaling the voltage of the task to the next level.
- Let $\Delta_2$ denotes the duration of the task after scaling the voltage of the task to another lower level.
- Let $v_1$ denotes the voltage of the task after scaling the voltage of the task to the next level.
- Let $v_2$ denotes the voltage of the task after scaling the voltage of the task to another lower level.
- Let $\sigma$ denotes the charge consumption of the initial profile before scaling the voltage of one of the tasks.
- Let $\sigma_1$ denotes the charge consumption of the profile after scaling the voltage of the task to the next level.
- Let $\sigma_2$ denotes the charge consumption of the profile after scaling the voltage of the task to another lower level.

The following equations can be obtained:

$$\frac{\Delta_1}{\Delta} = \frac{v}{v_1}$$  \hspace{1cm} (14)

$$\frac{\Delta_2}{\Delta} = \frac{v}{v_2}$$  \hspace{1cm} (15)

$$\frac{v_1}{v_2} = s \ (s > 1)$$  \hspace{1cm} (16)

The efficiency of scaling the voltage of the task to next level can be described as:

$$e_1 = \left| \frac{\sigma_1 - \sigma}{\Delta_1 - \Delta} \right|$$  \hspace{1cm} (17)

where $e_1$ is the efficiency of the profile obtained by scaling the voltage of the task to the next level. Another form of (17) can be obtained by replacing $\sigma$, $\sigma_1$ and $\Delta_1$ using (12) and (14).

$$e_1 = \left| \frac{kv^2 \Delta - kv^2 \Delta}{v_1 \Delta - \Delta} \right|$$  \hspace{1cm} (18)

Finally, (18) can be simplified as:

$$e_1 = kv v_1$$  \hspace{1cm} (19)

Then another form of (19) can be obtained by using (16) to replace $v_1$.

$$e_1 = ksv v_2$$  \hspace{1cm} (20)

Since $s > 1$,

$$e_1 > kv v_2 = e_2$$  \hspace{1cm} (21)

We can easily find that the right term in (21) is equal to $e_2$ ($e_2$ is obtained in the same way compared to $e_1$ as shown in (19)). (21) can be expressed as:

$$e_1 > e_2$$  \hspace{1cm} (22)

Therefore, the conclusion that for a given set of $n$ independent tasks, scaling the voltage of the tasks to the next level is more effective than to any other levels is proved.

**Theorem 2**: [The most effective theorem of decreasing the one-dimensional voltage to the next lower level] For a given task $\tau_i$, if it can be executed in $[S', F'] \subseteq [S, F]$, ($S'$ and $D'$ are the start time and end time of $\tau_i$, respectively, $S$ and $D$ are the start time and the end time of the task set, respectively). The consumption charge of $\tau_i$ executed in one-dimensional voltage (i.e., $\tau_i$ is executed in only one voltage) is less than the total consumption charge of $\tau_i$ executed in multidimensional voltages (i.e., $\tau_i$ is executed in multiple voltages) in $[S', F']$.

**Proof**: Inductive method is used to prove this theorem. Let $v$ and $\Delta$ denote the execute voltage and time of the task executed in one-dimensional voltage respectively.

Firstly, assume that there is a two-dimensional voltage set $V' = \{v_1, v_2\}$. Let $\Delta_1$ and $\Delta_2$ denote the needed execution time when $\tau_i$ is executed in the one-dimensional voltage of $v_1$ and $v_2$, respectively. It is obvious that $v_1$ and $v_2$ can not be equal. In this proof, (12) and (13) are also satisfied. We assume $\Delta_1$ and $\Delta_2$ are the durations of $v_1$ and $v_2$ when releasing task $\tau_i$ in two-dimensional voltages.

Since $\tau_i$ is ended at $D'$, the equation below is satisfied.

$$\Delta = \Delta_1' + \Delta_2'$$  \hspace{1cm} (23)

Then, from the execution code point of view, we assume the total code specifications of task $\tau_i$ is 1. Therefore, the code executed by $v_1$ and $v_2$ in $\Delta_1$ and $\Delta_2$ are $\Delta_1/\Delta_1$ and $\Delta_2/\Delta_2$, respectively.

Since $\tau_i$ is just finished in $\Delta_1' + \Delta_2'$, therefore,

$$\frac{\Delta_1'}{\Delta_1} + \frac{\Delta_2'}{\Delta_2} = 1$$  \hspace{1cm} (24)
Then we replace $\Delta_1$ and $\Delta_2$ by using (14) and (15). Another form of (24) can be obtained:

$$\Delta_1 v_1 + \Delta_2 v_2 = \Delta v$$  \hspace{1cm} (25)

By putting (23) and (25) together and solve the equations. The equations below are obtained.

$$\begin{cases}
\Delta_1' = \Delta \times (v_2 - v) / (v_2 - v_1) \\
\Delta_2' = \Delta \times (v - v_1) / (v_2 - v_1)
\end{cases}$$  \hspace{1cm} (26)

From the above equations, we can see if $v_1 < v_2$, then $v_1 < v < v_2$. If $v_1 > v_2$, then $v_1 > v > v_2$. Let $\sigma(v_1, v_2)$ denotes the charge consumption when $\tau_i$ executed in two-dimensional voltages. So we can get the expression of $\sigma(v_1, v_2)$.

$$\sigma(v_1, v_2) = k v_1^2 \Delta_1' + k v_2^2 \Delta_2'$$  \hspace{1cm} (27)

By replacing $\Delta_1'$ and $\Delta_2'$ using (26), (27) is redeclared as

$$\sigma(v_1, v_2) = k \Delta \left( v_1^2 \times (v_2 - v) / (v_2 - v_1) + v_2^2 \times (v - v_1) / (v_2 - v_1) \right)$$  \hspace{1cm} (28)

We take partial derivative of $\sigma(v_1, v_2)$ with respect to $v_1$ and $v_2$, respectively.

$$\frac{\partial \sigma(v_1, v_2)}{\partial v_1} = k \Delta \left( v_1^2 \times (v - v_2) / (v_2 - v_1) \right)$$

$$\frac{\partial \sigma(v_1, v_2)}{\partial v_2} = k \Delta \left( v_2^2 \times (v_1 - v) / (v_2 - v_1) \right)$$  \hspace{1cm} (29)

It is obvious that $\sigma(v_1, v_2)$ reach its minimum value when

$$\frac{\partial \sigma(v_1, v_2)}{\partial v_1} = \frac{\partial \sigma(v_1, v_2)}{\partial v_2} = 0.$$  

And the unique solution is $v_1 = v_2 = v$. This solution shows us that the charge consumption of $\tau_i$ executed in one-dimensional voltage is less than the total charge consumption of $\tau_i$ executed in two-dimensional voltages.

Secondly, we prove that task $\tau_i$ is executed in multidimensional voltages.

Assume that when $\tau_i$ is executed in $N - 1$ dimensional voltages, theorem 2 is true. Let $V_N = \{ v_1, v_2, \ldots, v_{N-1}, v_N \}$ and $\Delta_N = \{ \Delta_1, \Delta_2, \ldots, \Delta_{N-1}, \Delta_N \}$ denote the voltage range over set of task $\tau_i$ and the duration set corresponding to the voltage set, respectively. The total charge consumption can be expressed as:

$$\sigma(v_1, v_2, \ldots, v_{N-1}, v_N) = \sum_{1 \leq i \leq N} k v_i^2 \Delta_i'$$  \hspace{1cm} (30)

$$\sum_{1 \leq i \leq N} k v_i^2 \Delta_i' + k v_N^2 \Delta_N'$$  \hspace{1cm} (30')

Assume there is an equivalent voltage $\tau$ that charge consumed under it is equal to the total energy consumed under $v_i (i = 1, 2, \ldots, N - 1)$ in $\sum_{1 \leq i \leq N} \Delta_i'$ duration the execution of task $\tau_i$. Therefore,

$$\sum_{1 \leq i \leq N-1} \Delta_i' v_i = \tau \times ( \sum_{1 \leq i \leq N-1} \Delta_i' ) = \tau \times \Delta$$  \hspace{1cm} (31)

We can know from the assumption that the charge consumption of $\tau_i$ executed in $N - 1$ dimensions is larger than $\tau_i$ executed in one-dimensional voltage $v$. It can be expressed as:

$$\sum_{1 \leq i \leq N-1} k v_i^2 \Delta_i' \geq k \pi^2 \Delta$$  \hspace{1cm} (32)

Therefore, we regard $N$-dimensional voltages set $V_N'$ as two-dimensional voltages set $V_2' = \{ \tau, v_N \}$. So the equation below is satisfied.

$$\Delta \times \tau + \Delta_N' v_N = v \Delta$$  \hspace{1cm} (33)

According to the proved two-dimensional voltage situation, we can know:

$$k \pi^2 \Delta + k v_N^2 \Delta_N' > k v^2 \Delta$$  \hspace{1cm} (34)

Synthesize (33) and (34) we can get:

$$\sum_{1 \leq i \leq N-1} k v_i^2 \Delta_i' > k \pi^2 \Delta + k v_N^2 \Delta_N' > k v^2 \Delta$$  \hspace{1cm} (35)

It is obvious that when task $\tau_i$ is executed in $N$ dimension voltages, theorem 2 is true.

From the above proof, the charge consumption of $\tau_i$ executed in one-dimensional voltage is less than the total charge consumption of $\tau_i$ executed in multidimensional voltages.

To make the algorithms integrated, some constrains must be considered:

- None of the task could be finished after their deadline. $t_i + \Delta_i \leq d_i$ for all $i \in \tau$ (36)

where $d_i$ is the deadline of the $i$th task.

- All of the tasks must be finished before observation time $T$:

$$\sum_{i=1}^{n} \Delta_i \leq T$$  \hspace{1cm} (37)

- The parameter $\alpha$ must be larger than the objective function $\sigma_T$ at observation time $T$. (endurance constraint[1]).

$$\sum_{i=1}^{m} I_i F(T, min(t_i, d_i), min(t_i, d_i) + \Delta_i, \beta) \leq \alpha$$  \hspace{1cm} (38)

V. BATTERY-AWARE VOLTAGE SCHEDULING

A battery-aware voltage scheduling problem is $NP$-hard, even if tasks have the fixed-priorities [16], which implies that no optimal polynomial-time algorithm is likely to exist. In this paper, we propose two algorithms based on heuristics and GA can obtain energy-efficient voltage scheduling.

Lower power system can be obtained by using the slack time of the task set efficiently. From theorem 1 and theorem 2, scaling down the voltage to the next level and making task executed in only one-dimension voltage are more effective. This observation justifies the reduction of the speed of the
most effective task by one level at each time. Therefore our goal is to obtain a largest absolute value of gradient when we scale down the voltage of the tasks, so that we can get the largest energy saving in each task scheduling. The gradient of the function can be got by (17).

Although (17) had been applied in [12], the experimental results showed that they have considered the energy consumption of a single task only as shown in (39). It should be noted that our proposed algorithms focus on the total number of tasks.

\[
\frac{\sigma_i' - \sigma_i}{\Delta_i} = \Delta_i
\]  
(39)

A. Task Scheduling with Voltage Scaling for Uniprocessor System

1) Greedy Heuristic Algorithm: Our greedy heuristic algorithm obtains a more effective profile whose energy saving is remarkable compared to the previous research. Two phases are conducted in the proposed greedy heuristic algorithm.

Phase 1: Get a permissible schedule.
- Generate the initial task sequence by using the EDF algorithm with the highest voltage value that meets all timing constraints.
- Repeatedly scale down the voltage of the failed task or the tasks before them within the endurance constraint satisfied to repair the battery failure. To repair the failing tasks more effectively we using the least slack time to repair the failing tasks by using (17).

Phase 2: Distribute slack time.
- Scale down the speed of the tasks using (17) for the sake of using slack time to reduce the cost of energy without violating the deadline constraints;
- Swap the order of the tasks every time after scaling down the speed of one of the tasks until no task can be swapped any more;
- Output the final profile when the deadline constraints are violated or none of the task can be scaled.

Fig. 3 shows the relationship between energy saving and slack time using for the tasks.

2) DVS Algorithm Based on GA: In this paper, we propose a GA for static scheduling based on the following advantages.
- GAs have broad applicability and are intrinsically suitable for parallel implementation.
- In certain cases, GAs can find the global optimum of a problem with very high probability.
- Their performance is robust in many settings.

GA is an example of the meta-heuristics which have been successfully applied to a variety of problems. Since the search space is large and has a complex structure, finding a solution by genetic search is appropriate. Our GA requires the definition of a set of genetic operations and an evaluation function as follows:
- Coding scheduling individual;
- For uniprocessor system, we need not consider task assignment in the processors. Let \( \tau = \{ \tau_1, \cdots, \tau_n \} \) denotes a task set consist of \( n \) tasks in uniprocessor system. Each task \( \tau_i \) has the start time \( S_\tau \) and end time \( F_\tau \), \( V = \{ v_1, \cdots, v_5 \} \) denotes the voltage rang set which means different voltages are assigned to the tasks.
- Evaluation and optimization strategy;
- Our evaluation function captures the “degree of schedule and energy consumption” for a certain individual. The evaluation result is used as an individual fitness value. For a given application, decreasing individual cost value means that optimization result of this individual is good. We define the fitness function as follow:

\[
F = \begin{cases} 
C, & \text{Constraints are satisfied} \\
\frac{\sigma_T}{\sigma_T}, & \text{Constraints are not satisfied} 
\end{cases}
\]  
(40)

where C is a constant (We define the value of C a greater number of integer than battery parameter \( \alpha \). We set it to 100000 in this paper) and \( \sigma_T \) is the energy consumption of the individual at the observation time of \( T \). This formula demonstrates that, when the constraint conditions are satisfied, the fitness function is defined as power consumption of the individual. However, when constrains do not satisfied, the value of fitness function of the individual is added constant \( C \) as more penalty. Assuming no battery failure, we just want to obtain the smallest power consumption of the individual.
- Generating initial generation;
- Individuals are generated randomly as many as population size \( m \). Individual value is calculated by the evaluation function (5). In the case the same individual is generated, it is
deleted using differentiation of the same individual and a new different individual is obtained using mutation. Then the generation will be formed by sorting individuals in ascending order.

- Selection;
  As described above, all individuals in the population are sorted out according to their fitness functions, so the first individual is the best in this generation. The following operation is performed.

In our method, the top \( N \) (\( N \) denotes the number of the new individuals and \( N = m/4 \)) individuals are selected as the next generation of individuals.

- Recording the best individual;
  Because the fitness functions of his parents are the lowest in previous generation and if the minimal fitness value of individuals in the next generation is larger than its parents, minimal fitness of parent individual will be copied to his offspring.

- Crossover and mutation;
  The crossover operation is held between two parents (e.g., \( I_i, I_j \)). We randomly select two tasks (e.g., \( \tau_k \) and \( \tau_l \)) in \( I_i \) and \( I_j \), respectively. The voltages of the selected tasks \( \tau_k \) in \( I_i \) is exchanged with that of the selected task \( \tau_l \) of \( I_j \). The same operation is conducted twice. Therefore, we get four new individuals from two parents after crossover operator. A new individual is the best in this generation. The following operation is performed.

After crossover, we sort the individuals. If the same individual is found, then we operate mutation to it in order to generate a new different individual. We randomly select two tasks number in the repetitive individual. Then the two voltages of the tasks are changed by two voltages selected from the voltage set \( V \) randomly. It should be noted that the new voltages should be different from the preceding voltages.

- Confirming the precedence relation amongst the tasks;
  If the tasks have precedence relation among them, we need confirm it after all operation. When the relationship between the two tasks is not satisfied, we exchange the order of the two tasks. Each precedence relation is confirmed for one time, otherwise the program may be run into endless loop.

- Terminating condition.
  Repeat above processes until the value of the fitness function of the best individual do not changed in 200 loops.

B. Task Scheduling with Voltage Scaling for Periodic Tasks

In a conventional scheduling problems, a feasible schedule for periodic tasks exists if and only if there exists a feasible schedule for its hyperperiod, where the hyperperiod is the least common multiple (LCM) of the periods of all tasks in the task set. Due to the non-linearity of the battery behavior, the simple repetition of the optimal profile for hyperperiod is not optimal. For different hyperperiods, the optimal profile is not always the same.

1) Greedy Heuristic Algorithm for Periodic Tasks: In this section, our greedy algorithm also has two phases.

Phase 1: Generate a feasible schedule; We use non-increasing algorithm to generate a feasible list within hyperperiod without violating the deadline constraint for all tasks. Then failure repairing is also required in this section.

Phase 2: Distribute the slack time. The work in this phase are the same as shown in phase 2 of the task schedule for uniprocessor systems.

2) DVS Algorithm Based on GA for Periodic Tasks: GA-DVS for periodic tasks is almost the same algorithm for uniprocessor system. Profiles for hyperperiod in each individual are randomly generated. Then the individual fitness function value is calculated by (40).

C. Task Scheduling with Voltage Scaling for Multiprocessor Systems

1) Multiprocessor System Configuration: The system configuration for the DVS based multiprocessor system is the same as detail in [1]. A single battery drives multiple processors. Each processor has a dedicated DC-DC converter connected to it. As above assumption, there are \( M \) processors in the multiprocessor system, \( V_{\text{batt}} \) and \( I_{\text{batt}} \) represent the battery voltage and battery current respectively. \( v_{\text{proc}}(i) \) and \( I_{\text{batt}}(i)(i = 1, \cdots, M) \) are the voltage set and current set assigned for every processor, respectively. Therefore, \( I_{\text{batt}} = \sum_{i=1}^{M} I_{\text{batt}}(i) \).

2) Greedy Heuristic Algorithm for Multiprocessor System:
The greedy heuristic scheduling algorithm for multiprocessor system is an extension of the algorithm for the uniprocessor system. Different from uniprocessor system, the tasks are assigned to the processing elements according to a battery-aware priority function for multiprocessor system. If there are the precedence relation among the tasks, it should be considered the task process order for every processor.

Phase 1: Generate scheduling tables for each processing element;

The scheduling tables are generated by assigning the tasks to the processing elements under the precedence relation among the tasks without violating the deadline constraint (first priority) and non-increasing order (second priority). Note that in the processor assignment, inter-processor communication is avoided as much as possible. This might lead to some processing elements being active most of the time and others being relatively idle. Moreover, \( I_{\text{batt}} \) is the sum of the loads assigned to each processor. Other parts of processes are the same as in uniprocessor system.

Phase 2: Distribute slack time.

Distribute the slack time among the tasks in every processor efficiently under the constraint of non-increasing, which result in the sharpest slope of the battery load, the sharpest load profile result in the optimum consumption of the power [1]. The tasks in every processor are scaled until there are no slack time available or no more tasks can be scaled.

3) DVS Algorithm Based on GA for Multiprocessor System:
According to multiprocessor system architecture, GA-DVS has been changed accordingly in steps of fitness function, generating initial individuals and crossover operation.
Suppose \( \tau_k^i (i = 1, \ldots, M; k = 1, \ldots, s) \) is the task set assigned to \( k \)th processor. We define \( M \) strings are included in individual to indicate the sub-scheduling tables of every processor and \( M \) sub-fitness function values \((i.e., f_i (i = 1, \ldots, M))\) to evaluate every sub-scheduling tables. The fitness function value of the individual is obtained by adding the values of the sub-fitness function values \((i.e., f = \sum_{i=1}^{M} f_i)\). Each sub-fitness function is defined in the same way as shown in uniprocessor system. In this paper, we assume that there are two processors in multiprocessor system.

In the third step of GA-DVS, we randomly generate a quantity of \( s \) \((s\) is range from 1 to the half of the maximum number \( n \) of the tasks) tasks for the first processor \((P_1)\) and \( n - s \) for the second processor \((P_2)\). The new individuals are selected as the same way as shown in uniprocessor system.

The crossover operation is held between two parents. As described above, all individuals in the population are sorted out according to their fitness in ascending order. We select an individual according to the ascending order in generation as one parent \((e.g., I_k)\). We randomly select another individual \((e.g., I_l)\) as another parent. Let \( \tau_k^i \) \((k = 1, \ldots, s; i = 1, \ldots, m)\) denotes the \( k \)th task \((i.e., \tau_k^i)\) of the \( i \)th processor. Then we randomly select one task \( \tau_k^i \) in the \( I_k \). The voltage of the selected task is exchanged with the voltage in the corresponding position of task of \( I_l \) processor. The same operation is held for another processor \((p_2)\) of \( I_k \) and \( I_l \). Therefore there are four new individuals generated after the crossover operation. The crossover operations are conducted for the remainder individuals. If there are precedence relation among the tasks, the precedence relation are confirmed in the same way in the uniprocessor systems. We choose the population size \( m \) of individuals as the next generation and remove the excess of the individuals.

D. The complexity analysis of the genetic algorithm

We define \( T_{fitness}, T_{operation} \) are the computing time of the fitness of each individual and its operation time including crossover and mutation operation, respectively. Suppose that \( R \) is the number of individual in one generation, the iteration number of the genetic algorithm is \( I_{GA} \). The time complexity of algorithm is as follow:

\[
T_{GA} \approx I_{GA} \times TotalTaskNum \times (T_{operation} + T_{fitness})
\]  

(41)

Actually, the complexities of the genetic algorithms are distinct due to different programmer who write different details of the genetic algorithm.

One task in the individual is make up of a fixed number of elements \((e.g., current, voltage)\). ElementNum is defined as the maximum space of the task. Therefore, the space complexity of the genetic algorithm is \(O(R \times TotalTaskNum \times ElementNum)\).

VI. EXPERIMENTAL RESULTS

This section presents three sets of experiments demonstrating the effectiveness of our approaches. One class of examinations was used as the same data whose profiles were generated by a real-time application running on ITSY. One class of experimental data of task was generated randomly. Furthermore, the other class of experimental data was acquired by measuring practical application running on W90P710. By this way, the experiment result contains theoretical and practical data. The two developed algorithms are implemented using C language executed on a 3.00GHz Intel(R) Duo processor with a 2G RAM. We have conducted a comparison with the other algorithm \([1]\). For maximum flexibility of scheduling, we assume the arrival time of all the tasks to be 0 in all the experiments.

A. Comparative Examples

1) Task Scheduling with Voltage Scaling for Aperiodic Tasks on Uniprocessor Systems: The current profiles are generated by the application running on ITSY are shown in Table I. Four task graphs (Case 1, Case 2, Case 3, Case 4) are illustrated in Fig. 4. Case 1 presents the task set with dependencies which lead the current profile to be an increasing profile, Case 2 presents the task set with no dependencies, and case 3 and case 4 represented task graph generated randomly. To reveal the advantages of our algorithms in energy saving in this section, we list the comparisons of our algorithms and non-increasing algorithm \([1]\) for every cases.

![Fig. 4. Aperiodic task graphs for uniprocessor systems. (Nodes equipped with [A,B]:A is the execution time and B is deadline)](image-url)
Table II shows the experimental results of aperiodic tasks on uniprocessor systems using three algorithms. It is clear that the performance of our proposed algorithms are superior compared to non-increasing algorithm in terms of energy saving. The Greedy heuristic and GA-DVS were approximately 6.69% and 9.2% superior on average and 13.1% and 19% at the maximum, respectively. Table II illustrates that GA-DVS is the most accurate algorithm among three algorithms.

2) Experimental Results for Periodic Tasks on Uniprocessor System: Two cases (case 1: dependent task sets with the same period, case 2: independent tasks with different periods) of periodic tasks for uniprocessor systems are included in Fig. 5. The performance of the two proposed algorithms and non-increasing algorithm in the first hyperperiod (LCM) is listed in Table III. The relation between the best fitness value and the generation number of Case 1 is shown in Fig. 6. Our greedy heuristic algorithm is about 4% superior in case 1 and 7.7% superior in case 2 in terms of energy consumption respectively. GA-DVS has the best performance with 8.7% improvement at maximum in case 2 compared to non-increasing approach.

3) Experimental Results for Aperiodic Tasks on Multiprocessor System: Fig. 7 illustrates the dependency and time constraints for three cases (case 1: independent task set, case 2 and case 3: dependent tasks generated randomly) on multiprocessor systems. Case 1 has the maximum freedom of slack time using due to its no dependency. We assume that the number of processors is two.

Table IV lists the total processor energy consumption. From Table IV, our greedy algorithm has a better performance compared to non-increasing approach with 16.8% improvement at maximum in case 1. GA-DVS still superior than non-increasing algorithm and the proposed greedy heuristic algorithm in terms of energy saving.

B. Randomized experiments

In this section, the factors (i.e., deadline, current, duration) of the tasks were randomly generated. We conducted tests on five groups of task sets and each group contains different number of tasks. 15 samples were generated randomly for each task set and the average consumption charge and the average schedulability ratio of all groups are illustrated in Fig. 8 and Fig. 9. A more detailed comparison of the performance of the individual algorithms show the proposed two algorithms have the well performance in both with respect to charge consumption and the schedulability ratio. It is clear that in
In this paper, we have analyzed the characteristics of the battery and proposed two static voltage scheduling algorithms for battery-powered DVS systems. Comparative experiment results show that the superior performance of the proposed algorithms over the previous algorithms in terms of energy consumption. The proposed greedy scheduling algorithm is efficient for aperiodic and periodic task sets on both uniprocessor and multiprocessor systems. Another heuristic algorithm GA-DVS performs even better in some larger task sets. Compared to the other algorithms, GA-DVS is superior in energy saving. However, the time cost of GA is the largest among the comparative algorithms. Finally, in practical application experiments, our algorithms still superior than the comparative algorithm in consumption charge. Through the above experiments, we can confirm the validity and efficient of our algorithms.

REFERENCES


