Exploiting Maximum Value and Energy Awareness in Real-Time Scheduling

Gangman Yi
Samsung Electronics Co., Ltd.
gangman.yi@gmail.com

Abstract

The maximum power budget on electronic devices is limited by the rechargeable electrochemical batteries, so the energy awareness and high performance has been an important issue for the embedded systems. We present a scheduling algorithm that maximizes the value for the multiple task sets by applying dynamically at run time to cope with changes in the execution environment. In order to apply the scheduling algorithm to each task set when it goes out from a queue, we modified REW-pack and REW-unpack algorithms, and also investigated the admission algorithm of a task set to decide whether or not a new task set is accepted to go into the system.

keywords:

1. Introduction

The performance targeted design makes an electronic device power hungry. Objectives of the recent power saving research are either to limit maximum power consumption or to achieve energy efficiency. In the embedded system, the maximum power budget is limited by the rechargeable electrochemical batteries, which have fixed size or weight for portability. Because most systems have a much higher computing rate than the average throughput that must be sustained, efficient energy utilization can save a tractable amount of spare energy.

The time-constrained applications run on the embedded system like cellular phones and digital camcorders. The application has a hard real time deadline for each component in the system to ensure reliability and safety. The components are generally heterogeneous. For example, the laptop computer has CPU subsystem, disk, and color display. Each component may have different supply voltage, minimum power requirement, and processing rate. The distribution of tasks of an application to the components can be reduced to a scheduling problem. Most real time scheduling algorithms aim for low power consumption and deadline satisfaction.

However, the power saving modes such as low power state and idle state degrade performance. So the scheduling schemes which adopt low power and deadline satisfaction can degrade the high priority application not the low priority application. Therefore, we try to find a solution of another constraint except deadline satisfaction and minimization of power consumption.

We use the concept of reward(value), which is used in multimedia and image and speech processing. The characteristic of these applications is that an approximate but timely result may be acceptable, although missing deadline can be destructive in the mission critical applications like power plants or airborne navigation systems.

Those three constraints (energy, deadline, and value) play key roles in the current embedded systems. An optimal scheme would allow the system to run the most valued applications without wasting the available energy while still meeting all deadlines.

Our goal is to maximize the values by applying dynamically at run time to cope with changes in the execution environment. We try to show that our scheduling achieve low power consumption although it’s not optimal in terms of power consumption.

This paper is structured as follows. In section 2, we described previous work. Section 3 explains problem definitions, the task model, and assumptions. In this section, we explain algorithms that maximize the system value. Section 4 presents the experiment description and
the environment for the simulation. In section 5, we interpret the experimental results obtained through simulation and conclude the paper in section 6.

2. Previous Work
Dynamic Voltage Scaling (DVS) is one of key techniques that reduce energy consumption in real-time scheduling. It achieves low energy consumption by abating the supply voltage and operating frequency of processors. Therefore, it is regarded as a trade-off between energy saving and increased execution time. [1] suggested the simultaneous voltage scaling algorithm for both processors and communication links in real-time distributed systems. If DVS is applied incautiously, DVS may not satisfy deadline guarantees. However, real-time DVS (RT-DVS) modifies the real-time scheduler and the task management in operating systems to provide significant energy savings while maintaining real-time deadline guarantees [13].

The well-known algorithm of real-time scheduling for low power consumption is the earliest deadline first (EDF) scheme. Dudani, Mueller and Zhu improved EDF algorithm using feedback mechanisms [4]. Their algorithm distributes the slack produced by the invocation of the task at a low frequency, which is determined by a feedback mechanism within the same invocation. Another low power algorithm is addressed through resource sharing among different configurations in [7]. The non-preemptive scheduling heuristic for low power is investigated in [8]. It used voltage scaling, task scheduling, and task allocation to minimize power consumption.

Low power scheduling algorithm is studied in multi-processor as well as single-processor [9]. Zhang et al.'s work is a two-phase framework that consists of task assignment, task ordering, and voltage selection. Another power-aware scheduling algorithm in multi-processor systems is suggested to distribute and share slacks between independent and dependent tasks [10]. Global Scheduling with Shared Slack Reclamation (GSSR) and List Scheduling with Shared Slack Reclamation (LSSR) gain considerable energy saving, when the task’s execution time is smaller than its worst case execution time. Additional benefit of GSSR and LSSR is the increase of the reliability in the system.

Time constraint is not an only factor to be considered in low power scheduling. In [11, 14], the concept of value is introduced in order to emphasize not only performance but also energy consumption in real-time embedded systems. REW-pack and REW-unpack algorithms give a solution to select tasks to execute and determine each selected task’s speed level on a variable voltage processor. Thus, the total value of the system, which is defined as the sum of selected task values for execution, is maximized without violating the timing and energy constraints. This scheduling algorithm is static, since it finds out the subset of tasks to be scheduled before execution. Quan and Hu’s static scheduling scheme determines the optimal voltage of a variable voltage processor and then find a power efficient fixed-priority schedule [12].

In this paper, we propose a scheduling algorithm to maximize the value for the multiple task sets, while [11, 14] deals with a single task set. Our proposed scheduling algorithm is dynamic, since each task set arrives at the system based on an exponential distribution. Scheduling of a new task set must be done when a task set in the system completes its execution. Therefore, the schedule of a task set running in a system affects the schedule of task sets in a queue that wait for execution, since the slack of task sets in a queue is reduced. In other words, the time budget of a new task set (slack) is decremented by the amount of the difference between the arrival time of a task set in a queue and the end time of execution of a task set running in a system. In an extreme case, the task set in a queue cannot be admitted in the system, since the reduced slack is too small although it is executed at the highest voltage.

In order to apply the scheduling algorithm to each task set when it goes out from a queue, we modified algorithms (REW-pack and REW-unpack) proposed in [11, 14]. We also investigate the admission algorithm of a task set to decide whether a new task set is accepted to go into the system or not. Before going into the detail of the algorithm, we define a problem and present some notations in the next section.

3. Problem Definition: Task Model
The goal of the typical real-time system scheduling is to meet deadline and to reduce power consumption. This makes less important tasks that consume much less power selected, instead of important tasks to decrease power consumption and to meet deadline. Time and energy constraints are not sufficient criteria to determine which tasks are added or dropped. The value is used in order to tackle this problem [11]. Value function assigns values to the tasks, as a function of the amount of time the task is allotted. In other words, the more the CPU is
used by a certain application, the more value it accrues. Tasks’ property (mandatory or optional) is also considered when tasks are determined to be dropped. Tasks’ property keeps mandatory tasks from dropping.

To deal with task sets with various arriving time in multi-task system, admission algorithm for new arriving task set is necessary. When a new arriving task set requests admission to the system, the system determines if it can be processed by its deadline without causing existing task sets to violate their deadline.

We assume that a single task set is active in a single processor system. The tasks in a task set are serialized and a task can be executed only after its all preceding tasks finish their execution. It is not a requirement that all tasks of a given task must be scheduled. A task is categorized into a mandatory task and an optional task. Mandatory tasks have to be executed, but optional tasks can be dropped when the task set that has tasks is injected to the system. For example, if a task set \( \{ T_1, T_2, T_3 \} \) has two mandatory tasks \( (T_1 \text{ and } T_3) \), and one optional task \( (T_2) \), the execution result can be either \( T_1-T_2-T_3 \), or \( T_1-T_3 \).

The tasks are to be executed on a variable voltage processing element (PE) with the ability to dynamically adjust its frequency and voltage on application requests. There are \( L \) available voltages \( \{ v_1, v_2, \ldots, v_L \} \) and the given voltage can be applied only some frequency range. Each task can run at any of the available speeds and we say that a task runs at speed level \( j \) if the speed of the task is set to \( v_j \).

We also assume that the task execution time and energy consumption at different voltage level are known for all tasks and all voltage level. The following notations will be used for the next description.

\( S_k \) : task set in a distributed system \( (1 \leq k \leq N) \)
\( T_i^k \) : \( i \)th task of the task set \( S_k \) \( (1 \leq i \leq M_k) \)
\( D^k \) : deadline(completion time) of the task set \( S_k \)
\( v_j \) : voltage level \( (1 \leq j \leq L) \)
\( e(T_i^k, v_j) \) : consumed energy for task \( T_i^k \) at voltage level \( v_j \)
\( r(T_i^k, v_j) \) : execution time for task \( T_i^k \) at voltage level \( v_j \)
\( r(T_i^k) \) : value, the number of CPU cycles required for task \( T_i^k \)
\( E_{\text{max}} \) : total energy budget in the system

Our objective can be divided an inter-task domain and an intra-task domain. In an intra-task domain, we aim to maximize the system value that means sum of value of tasks to be scheduled, through scaling voltage and dropping optional tasks within a task set. In an inter-task domain, we aim to satisfy schedulibility, which guarantees deadline satisfaction of a task set. When a new task set wants to enter into the system, the system must determine whether the new task set can be processed by their deadlines, without either violating the guaranteed deadlines to the existing task sets or not exceeding the total energy imposed by the amount of energy available in the system.

Thus, the problem is to find a schedule of the given task sets and each voltage level associated to each task in the task set. The result of schedule determines which optional tasks join to schedule and voltage level of both mandatory and optional tasks.

When we decide task set admission and select a proper scheduling algorithm, we use the following two properties of a task set.

\( m(k) \) : sum of execution time of only mandatory tasks in the task set \( S_k \) at the lowest voltage level, \( v_1 \)

\( m(k) = \sum_{\text{all mandatory task}} t(T_i^k, v_1) \)

\( M(k) \) : sum of execution time of only mandatory tasks in the task set \( S_k \) at the highest voltage level, \( v_L \)

\( M(k) = \sum_{\text{all mandatory task}} t(T_i^k, v_L) \)

If \( m(k) \) is smaller than the deadline, all mandatory tasks can complete their execution at the lowest voltage until the deadline. If the voltage of some mandatory tasks increases, the total execution time of them is reduced. It means that some optional tasks may join the schedule. Therefore, the total value increases at the expense of
extra consumed energy incurred by both voltage scaling and optional task execution.

If $m(k)$ is larger than a deadline and $M(k)$ is smaller than a deadline, all mandatory tasks can complete their execution at the highest voltage until the deadline. However, the consumed energy of all mandatory tasks is maximum, and total value is not maximum since no optional tasks are in the schedule. To decrease consumed energy and increase the total value, we can decrease the voltage of some mandatory tasks and insert optional tasks into the schedule, while still meeting the deadline.

$M(k)$ is regarded as the lower bound for schedulibility. Since all mandatory tasks run at the fastest level, the task set must be rejected if the deadline is less than $M(k)$.

In section 4 and section 5, we use $m(k)$ and $M(k)$ to explain task set admission and scheduling algorithm.

4. Task Set Admission
A slack for a task set can vary, since a task set arrives before some task set finishes its execution. In the worst case, it happens that there is no slack of a task set.

A task set that comes out from a system queue goes into the system, if it is guaranteed that its execution can complete until the available time budget. Otherwise, it is denied. This decision is made by the comparison with $M()$. If a deadline of a task set is less than $M()$, it means that a task set must be denied.

5. Value Maximization Algorithm
After a task set is decided to enter into a system, we can choose which scheduling algorithm will be applied to it. The following two conditions are tested. We assume that $S_n$ is a task set running in the system and $S_{n+1}$ is a pending task set to schedule.

**Condition 1:**

$\text{end\_time}(S_n) + m(n+1) \leq \text{arrival\_time}(S_{n+1}) + D_{n+1}$

**Condition 2:**

$\text{end\_time}(S_n) + M(n+1) \leq \text{arrival\_time}(S_{n+1}) + D_{n+1}$

When $S_{n+1}$ satisfies Condition 1, the modified REW-pack algorithm is applied to schedule a task set $S_{n+1}$. When it satisfies Condition 2, the modified REW-unpack algorithm is used.

Figure 1 shows the flow of the modified REW-pack algorithm. There are three major components in the algorithm; add-task, increase-speed and drop-task. All mandatory tasks are included in an initial schedule before the add-task procedure.

![Flow of Modified REW-pack Algorithm](image)

The modified REW-unpack algorithm is similar to REW-pack algorithm. There are also three major components that are add-task, decrease-speed, and drop-task. All mandatory tasks are selected before any other procedure like modified REW-pack algorithm.

Before applying the scheduling algorithm, we should recalculate the available time budget (the adjusted deadline). The adjusted deadline is denoted as following.

$D_{n+1} = D_{n+1} - (\text{end\_time}(S_n) - \text{arrival\_time}(S_{n+1}))$

**Add-task**
The task is allowed to add to schedule when it satisfies below conditions.

- The task was not checked yet.
- Total execution time including a task $\leq \hat{D}$
1. Initialize: Selected(i)=false; consider(i)=false; all i = {1,2,3,… N}
energy=0; time=0; SR=0; R=0;

2. If task_character(i) = Mandatory
   a. sol_selected(i)=selected(i);
      sol_speed(i)=speed(i); all i = {1,2,3,… N}
   b. SR=R;
3. if (\exists i, consider(i)=false or time>D) do
   a. i= add_task();
   b. if I != -1
      i. selected(i)=true; considered(i)=true;
         energy=energy+e(i,1); time= time+ t(i,1); R=R+r(i);
      ii. go to step 2.
   c. i= increase_speed()
   d. if i!=-1
      i. energy=energy + e(i, speed(i)+1);
         time = time + t(i, speed(i)+1)- t(i, speed(i));
         speed(i) = speed(i)+1;
      ii. go to step 2
   e. i= drop_task();
   f. energy=energy-e(i, speed(i));
time = time - t(i, speed(i)); R = R - r(i); select d(i) = false; g. Go to step 2
4. Return solution (sol_selected, sol_speed, SR)

Figure 2. Modified REW-pack Algorithm

 Modified REW-unpack Algorithm

 1. Initialize: selected(i) = false; considered(i) = false all i = {1, 2, 3, ..., N}
     Energy = 0; time = 0; SR = 0; R = 0;
 2. if character(i) = Mandatory
     a. sol_selected(i) = selected(i); sol_speed(i) = speed(i);
     b. SR = R;
 3. if energy <= E_max and SR < R
     a. sol_selected(i) = selected(i); sol_speed(i) = speed(i). all i = {1, 2, 3, ..., N}
     b. SR = R;
 4. if (\exists i, considered(i) = false) or (energy > E_max) do
    a. i = add_task()
    b. if i = -1
       i. selected(i) = true; considered(i) = true; energy = energy + e(i, M);
       speed(i) = M; time = time + t(i, M); R = R + r(i)
       ii. Go to step 2.
    c. i = decrease_speed()
    d. if i = -1
       i. energy = energy - e(i, speed(i) - 1) - e(i, speed(d(i)));
       time = time + t(i, speed(i)); speed(i) = speed(d(i) - 1)
       ii. Go to step 2.
    e. i = drop_task()
    f. energy = energy - e(i, speed(i)); time = time + t(i, speed(i))
       R = R - r(i); selected(i) = false
    g. Go to step 2
5. Return solution (sol_selection, sol_speed, SR)

Figure 3. Modified REW-unpack Algorithm

6. Example (Task Set Admission & Scheduling Algorithm Selection)

Case 1 (Task set admission, Modified REW-pack algorithm selection)

Let’s assume that there are two task sets (S_1, S_2) in the system and the current time is t. Assume that m(2) and M(2) are 6, 1.8 respectively.

One task set (S_1) is executing in the system and will finish its execution at time t + 5. The other task set (S_2), whose deadline is 12, is generated at time t + 2. S_2 can not be executed immediately and must wait for 3 times, since S_1 is running in the system. S_2 must be finished by t + 4(=t + 2 + 12). Since this task set starts at time t + 5, it causes the deadline of S_2 to reduce to 9. S_2 can finish its execution, because m(2) is less than the adjusted deadline. Therefore, S_2 is admitted. Modified REW-pack algorithm is applied to schedule within S_2.

Case 2 (Task set admission, Modified REW-unpack algorithm selection)

If the deadline of S_2 is 8, the task set must finish at time by time t + 10(t + 2 + 8). Since this task set starts at time t + 5, the effective time budget of this task set is 5. Therefore, this task set is admitted and use modified REW-unpack algorithm, since this adjusted deadline is less than m(2) and greater than M(2).

Case 3 (Task set rejection):

If the deadline of S_2 is 4, the adjusted deadline is 1. Since it is less than both m(2) and M(2), it is rejected.

7. Example (Scheduling Tasks Within a Task Set)

Once a task set is accepted, tasks within a task set are scheduled. Scheduling objective is to maximize system value. Two constraints of scheduling are time and energy. Before starting to schedule, the system already knows that which algorithm is used among modified REW-pack or modified REW-unpack.

Modified REW-pack algorithm uses voltage increasing of each task. In other hand, modified REW-unpack algorithm uses voltage decreasing of each task. To select the proper voltage level for each task, the benefit of voltage level transition of each task, \( \frac{\Delta E}{\Delta V} \), is a criterion.

To explain some specific example, we listed transition values of each task in the Table 2 and Table 3, respectively. The task set has 4 mandatory tasks (\( T_1^{\prime}, T_2^{\prime}, T_3^{\prime}, T_4^{\prime} \)) and 4 optional tasks (\( T_5^{\prime}, T_6^{\prime}, T_7^{\prime}, T_8^{\prime} \)). The system has 4 voltage levels.

<table>
<thead>
<tr>
<th>Task</th>
<th>V_1</th>
<th>V_2</th>
<th>V_3</th>
<th>V_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>0.38</td>
<td>0.3</td>
</tr>
<tr>
<td>2</td>
<td>1.5</td>
<td>0.75</td>
<td>0.56</td>
<td>0.45</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>1.25</td>
<td>0.94</td>
<td>0.76</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.5</td>
<td>0.38</td>
<td>0.3</td>
</tr>
</tbody>
</table>
### Table 1. Execution Time of Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Task Value</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50.01</td>
<td>150.01</td>
</tr>
<tr>
<td>2</td>
<td>132.41</td>
<td>201.25</td>
</tr>
<tr>
<td>3</td>
<td>203.7</td>
<td>231.6</td>
</tr>
<tr>
<td>4</td>
<td>178.34</td>
<td>207.35</td>
</tr>
<tr>
<td>5</td>
<td>132.27</td>
<td>187.64</td>
</tr>
<tr>
<td>6</td>
<td>141.22</td>
<td>187.64</td>
</tr>
</tbody>
</table>

### Table 2. Consumed Energy of Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Task Value</th>
<th>Consumed Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.24</td>
<td>2.83</td>
</tr>
<tr>
<td>2</td>
<td>13.62</td>
<td>2.7</td>
</tr>
<tr>
<td>3</td>
<td>15.03</td>
<td>3.29</td>
</tr>
<tr>
<td>4</td>
<td>10.63</td>
<td>3.86</td>
</tr>
<tr>
<td>5</td>
<td>9.57</td>
<td>3.5</td>
</tr>
<tr>
<td>6</td>
<td>8.47</td>
<td>3.12</td>
</tr>
<tr>
<td>7</td>
<td>14.51</td>
<td>5.13</td>
</tr>
<tr>
<td>8</td>
<td>11.97</td>
<td>4.3</td>
</tr>
</tbody>
</table>

### Table 3. $\Delta T / \Delta E$

<table>
<thead>
<tr>
<th>Task</th>
<th>Task Value</th>
<th>$\Delta T / \Delta E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>1.12</td>
</tr>
<tr>
<td>2</td>
<td>150</td>
<td>1.5</td>
</tr>
<tr>
<td>3</td>
<td>250</td>
<td>1.19</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>0.81</td>
</tr>
</tbody>
</table>

### Case 1 (Pack): Let’s assume that the task set deadline is 9 and the system energy budget is 900. In this case, we will show the example of the modified REW-pack algorithm application.

First, add_task() function is called to find the optional task to execute. The criterion function $A(T^k_i) = r(T^k_i) / (t(T^k_i, v_1) e(T^k_i, v_1))$ is applied to all optional tasks. Then $A(T^5_3) = 1.12$, $A(T^6_6) = 1.5$, $A(T^7_7) = 1.19$, and $A(T^8_8) = 0.81$. Optional task $T^6_6$ is selected, since it has maximum value and adding it to execution set does not exceed deadline and energy constraint. The optional task $T^7_7$ is selected at the next iteration.

After two iterations, increase_speed() function is called, since there is no room for adding a new optional task. The criterion function $\Delta T / \Delta E$ is applied to find a task to maximize its value. According to Table 3, $T^3_3$ is selected and the voltage level of its task is increased by one level.

At the next call of increase_speed(), $T^7_7$ is selected to increase its voltage level. Voltage increment of these two tasks makes total execution time reduced. So we can add another task $T^5_5$ to the schedule.

Then, we do not increase voltage and do not add an optional task anymore. The final schedule has 5 tasks ($T^1_1$, $T^2_2$, $T^3_3$, $T^4_4$, $T^5_5$), with the lowest voltage level ($v_1$) and 2 tasks ($T^3_3$, $T^7_7$) with the voltage level ($v_2$).

### Case 2 (Unpack): Let’s assume that the task set deadline is 9 and the system energy budget is 900. However, if a new task set arrives while other task set is running, its deadline must be adjusted. So the deadline of an adjusted task set is 5. In this case, we will show the example of the modified REW-unpack algorithm application.

1. Total execution time of task set is 7.75 and energy consumed is 564.64.
2. After addition of task $T^5_5$, total execution time and energy consumed are 8.25 and 733.56 respectively.
3. The total execution time, energy consumed, and system value are 8.25, 733.56, and 1200.
First, 4 mandatory tasks \( T_1^k, T_2^k, T_3^k, T_4^k \) are placed in the schedule and their initial voltage is the highest voltage \( v_4 \). This initial schedule’s total execution time and energy consumed are 1.8 and 899.18, respectively.

Since this initial schedule has no sufficient energy budget to add an optional task, the system must run decrease_speed() function to decrease the voltage level of mandatory tasks. The system chooses \( T_4^k \) to decrease its voltage level to \( v_3 \). The task selection criterion in decrease_speed() is \( \Delta E / \Delta T \), which is opposite to the criterion of increase_speed(). System needs to call decrease_speed() function several times to get sufficient energy to add optional tasks. \( T_2^k \) (2 times), \( T_3^k \) (2 times), and \( T_4^k \) (2 times) are selected to decrease voltage level. After that, the optional task \( T_5^k \) is added to the schedule. These voltage decrement and addition of optional tasks generate the final schedule as \{ \( T_1^k - v_2 \), \( T_2^k - v_2 \), \( T_3^k - v_2 \), \( T_4^k - v_2 \), \( T_5^k - v_2 \), \( T_6^k - v_2 \), \( T_7^k - v_2 \) \}, where each pair denotes task and voltage level. The total execution time, consumed energy, and system value of the final schedule are 5,770.79, and 900, respectively.

8. Experimental Results
To validate our real-time scheduling algorithm, a simulator is developed using C. We executed simulation in Unix. To simulate the system environment, we used the simulation result of IBM PowerPC 405LP processor in [11]. Its speeds, voltages, and power ranges are shown in Table 5.

We generated task sets synthetically. The inter-arrival time between task sets follows an exponential distribution. The CPU cycles of a task (value) is randomly obtained in the range of [50, 250]. Table 6 shows the number of mandatory tasks and optional tasks of our test case. The execution time of each task was calculated as the number of CPU cycles divided by the frequency. The power of a task \( i \) at the voltage level \( j \) is computed as \( P_{i,j} = P_{j}^{\text{max}} + \alpha_i (P_{j}^{\text{max}} - P_{j}^{\text{min}}) \), where \( \alpha_i \) is in \([0, 1]\). The consumed energy can be calculated as a product of the execution time and power.

We compared our proposed scheduling algorithm with [11]. Our task model includes an optional task, although [11] has optional codes inside a task. Another difference is scheduling multiple task sets. Since a task set arrives at the system while another task set is running in the system, the runtime admission control of a task set is required. Figure 4 shows the difference of the proposed scheduling (Proposed) and [11] (Original) in terms of value, energy, and total execution time. The total value obtained from the proposed scheduling is larger than that of [11], although it consumed more energy. Since the total execution time is reduced, it means that we have more freedom for utilizing the slack of the next task set to be scheduled.

| Task set 1 | 3 | 6 | Pack |
| Task set 2 | 6 | 8 | Unpack |
| Task set 3 | 5 | 9 | Unpack |
| Task set 4 | 3 | 5 | Unpack |
| Task set 5 | 3 | 5 | Pack |

Table 6. The Number of Tasks in Each Task Set
9. Conclusion

The concept of value (reward) is used in some real-time applications such as multimedia, image, and speech processing. The characteristic of these applications is that an approximate but timely result may be acceptable, although missing deadline can be destructive in the mission critical applications like power plants or airborne navigation systems.

We modified the value maximization algorithm [11] to schedule mixed tasks, which is optional or mandatory. To schedule multiple task sets, task set admission control is necessary. However, our algorithm is not energy minimization scheduling algorithm. Its objective is not to exceed the energy constraint. We use either unpack or pack algorithm dynamically according to the time budget for a new task set, although [11] sticks to either of algorithms.

The simulation result shows that the schedule obtained by the proposed algorithm has more value than [11]. We need more experiments to verify our results. For the further work, we need to study about changing the voltage of a task in a running task set when a new task set arrives in order to minimize task set rejection.

10. REFERENCES


[3] W. Kwon, T. Kim, Optimal voltage allocation techniques for dynamically variable voltage


