An Iterative Algorithm for Battery-Aware Task Scheduling on Portable Computing Platforms*

Jawad Khan and Ranga Vemuri
ECECS Department, University of Cincinnati, Cincinnati, Ohio 45221-0030, USA
{jkhan,ranga}@ececs.uc.edu

Abstract

In this work we consider battery powered portable systems which either have Field Programmable Gate Arrays (FPGA) or voltage and frequency scalable processors as their main processing element. An application is modeled in the form of a precedence task graph at a coarse level of granularity. We assume that for each task in the task graph several unique design-points are available which correspond to different hardware implementations for FPGAs and different voltage-frequency combinations for processors. It is assumed that performance and total power consumption estimates for each design-point are available for any given portable platform, including the peripheral components such as memory and display power usage. We present an iterative heuristic algorithm which finds a sequence of tasks along with an appropriate design-point for each task, such that a deadline is met and the amount of battery energy used is as small as possible. A detailed illustrative example along with a case study of a real-world application of a robotic arm controller demonstrates the usefulness of our algorithm.

1. Introduction

Battery powered portable systems have finite amount of battery energy available and therefore battery lifetime maximization is one of the most important design goals for such systems. In this paper we present an iterative heuristic algorithm based on the battery discharge characteristics. Our goal is to meet a desired deadline and save as much battery energy as possible.

Target Hardware Architecture: The algorithm described in this work is applicable to any embedded platform, although the methods for changing the energy consumption of a task vary depending upon which processing element is used. In a processor based embedded system power-performance trade-offs can be achieved by voltage and clock scaling. It is assumed that several discrete voltage and frequency combinations are available. If the embedded platform has an FPGA as the main processing element then it is assumed that several different hardware implementations are available which can be downloaded in the form of bitstreams. It is assumed that for each design-point, performance and total power consumption estimates are available for any given portable platform, including the peripheral components such as memory and display power usage. Further, it is assumed that intertask communication occurs via shared memory and the energy cost and latency of the memory transfers are a part of the execution time and energy costs of the task under consideration.

Application Specification: The application is described as a directed acyclic task graph (DAG) \( G(V,E) \). The vertices (nodes) of the graph are tasks which are to be executed on the portable platform and each task \( v \in V \) has several different implementation options available called design-points. Associated with each task \( f \) and its design-point \( j \) is its execution time \( E_{f,j} \) and current consumption \( I_{f,j} \). The current consumption of a task is assumed to be measured as the average total current consumption of the portable platform which is the cumulative current consumption of all subsystems being used in the portable platform. The edges \( E \) describe data and control dependence between different tasks of the task graph. In the rest of the paper we will use \( n = |V| \), \( e = |E| \) and there are \( m \) design points available for each task. There is a deadline \( d \) associated with the task graph before which all the tasks must be completed.

Problem Description: Given a DAG \( G(V,E) \), a set of design-points for each task, execution time, current usage estimates for each design-point and a desired deadline for the completion of the task graph, determine a valid schedule which does not violate the control and data dependencies of the tasks in the task graph and also find a mapping of each task to a suitable design-point such that the deadline for the entire task-graph is met and the battery energy used is as small as possible. Our algorithm finds an initial sequence and then assigns design-points to the tasks. The suitability of the task sequence and the design-point selection is judged by using a battery model due to [2] which will be discussed in Section 3. The chosen task sequence is modified using a heuristic approach and design-point selection is performed again to improve the quality of the solution. The algorithm is described in detail in Section 4. We tested the algorithm using different task-graphs and design-points and the results are discussed in Section 5.

2. Related Work

In processor based embedded computing platforms dynamic voltage and frequency scaling has been proven to be extremely effective for low power execution of tasks [4][5][6][8]. In battery-powered embedded systems the energy source is non-linear therefore, the existing voltage scaling techniques are not directly applicable to these systems. Luo

---

* This work was sponsored in part by the Ohio Board of Regents PhD Enhancement Program
and Jha studied static task scheduling for battery powered multiprocessor environments [5]. They used a battery model which was based on Peukert’s law and an empirical model due to Pedram [6]. Rakhmatov et al developed an algorithm for battery-aware task scheduling using dynamic programming [1] along with its other variants. Chowdhury et al [7] proposed a simplified heuristic which reduced the voltage level of the tasks as much as possible starting from the last task in the schedule. In contrast to these works, our work provides a way to simultaneously solve the task sequencing and design-point assignment in an iterative fashion. In any given iteration a valid schedule and assignment is available which can be used. If the user wishes then the solution quality is improved over subsequent iterations. Further, compared to our algorithm, it is not easy to implement a Simulated Annealing or Linear Program Formulation based algorithms on an embedded computing platform which has inherent limitations on memory and battery capacity.

3. Battery Characteristics and Motivation

Rated Capacity of a battery is defined as the capacity of the battery (in mAh) under a nominal constant current discharge and is reported by the manufacturer. It is observed that higher rates of discharge tend to reduce the rated capacity significantly (rate capacity effect) and reducing discharge rates between heavy discharge periods allows the battery to regain some of its lost capacity (recovery effect) [3]. Rakhmatov et al. [2] developed a variable load analytical model based on the laws of chemical kinetics, which takes into account both the rate capacity effect and the recovery effect. Equation 1 describes the battery model.

\[
\sigma = \sum_{k=0}^{n-1} I_k \left( \Delta_k + 2 \sum_{m=1}^{10} \frac{e^{2m} \beta^2 m^2 (T-t_k) - e^{2m} \beta^2 m^2 (T-t_k)}{\beta^2 m^2} \right)
\]  

(1)

The value of \( \sigma \) gives the amount of charge lost by time \( T \), which is the length of a current discharge profile having \( n \) distinct discharge intervals. \( I_k \) is the current drawn from the battery in the \( k^{th} \) discharge interval, where \( t_k \) is the start time of the \( k^{th} \) discharge interval and \( \Delta_k \) is the duration of this interval. The battery lifetime is estimated by evaluating Equation 1 for increasing values of \( T \) and stopping where \( \sigma = \alpha \). At this point the value to \( T \) is taken as the battery lifetime. Equation 1 is used as the battery-aware cost function to be minimized. We have chosen to use this battery model because of its high accuracy and low computational complexity. It was shown in [1] that for a set of \( n \) tasks if dependencies are ignored and the value of \( \alpha \) is assumed to be sufficiently large then sequencing tasks in the non increasing order of their currents is the best and sequencing the tasks in the non decreasing order of their currents is the worst. This property is also important for task-graphs where dependencies are present because it provides the lower and upper bounds on the value of cost function given in Equation 1. The authors in [7] also proved that given a pair of two identical tasks in the profile and a delay slack to be utilized by down scaling, it is always better to use the slack on the later task than on earlier task. We use the above two properties along with the observation that tasks which have lower overall average energy consumption should be given priority for voltage down scaling.

4. Battery-Aware Task Sequencing and Design-Point Assignment

Some important definitions are presented below first:

Execution Time matrix (\( D \)) is an \((n \times m)\) matrix where \( D_{i,j} \) gives the execution time of task \( i \) using design-point \( j \); for each task \( i \), the execution times of the design-points are stored in ascending order of magnitude.

Current matrix (\( I \)) is an \((n \times m)\) matrix where \( I_{i,j} \) gives current of task \( i \) using design-point \( j \); for each task \( i \) the currents of the design-points are stored in descending order of magnitude.

Design-Point Selection matrix (\( S \)) is an \((n \times m)\) matrix where \( S_{i,j} \) is 1 if task \( i \) is assigned to design-point \( j \). \( S \) is initialized such that \( S_{i,j} = \{1 \text{ if } j = m, 0 \text{ otherwise} \} \).

Energy Vector \( E \): is a row vector where each element specifies a task and the tasks are stored in increasing order of their average energies.

Slack Ratio (\( SR \)) of a design-point is defined as the ratio of the amount of slack left to the deadline, if that design-point is chosen for execution. If \( r \) is the execution time of a design-point and \( d \) the deadline of the task graph then formally \( SR \) is defined as: \( SR = \frac{(d-r)}{d} \). Similarly, if \( SR \) is to be calculated for several design-points chosen, then \( r \) would be defined as the sum of the execution times of all the design-points chosen. \( SR \) gives an indication of how much slack is left which needs to be utilized. It is beneficial to use as much slack as possible. Therefore, a smaller value of \( SR \) is better.

Current Ratio (\( CR \)) of a design-point is defined as \( CR = \frac{(I-I_{\text{avg}})}{(I_{\text{max}}-I_{\text{min}})} \), where \( I \) is the average current used by the design-point and \( I_{\text{avg}}, I_{\text{max}} \) and \( I_{\text{min}} \) are the maximum and minimum currents among all the design-points of all tasks. \( CR \) is normalized to be between 0 and 1. \( CR \) gives an indication of relative current of a particular design-point when compared to all the other design-points. A smaller value of \( CR \) is better.

Energy Ratio (\( ENR \)) of a task sequence is defined as \( ENR = \frac{(E_n-E_{\text{avg}})}{(E_{\text{max}}-E_{\text{min}})} \), where \( E_n \) is the total average energy used by the chosen design-points for all the tasks. Energy ratio is low if a set of design-points uses lower overall average energy. Its value is between 0 and 1.

\[
E_n = \sum_{i=1}^{n-1} I_{i,c} \times V_{i,c} \times D_{i,c}
\]

\( I_{i,c} \) and \( V_{i,c} \) are the current and the voltage of the design-point \( c \) chosen for task \( i \), respectively, and \( D_{i,c} \) is its corresponding execution time. \( E_{\text{min}} \) and \( E_{\text{max}} \) are the energies of tasks sequences if all the lowest and highest power design-points are used for all tasks, respectively.

\[
E_{\text{min}} = \sum_{i=1}^{n-1} I_{i,\text{min}} \times D_{i,\text{min}} \quad E_{\text{max}} = \sum_{i=1}^{n-1} I_{i,\text{max}} \times D_{i,\text{max}}
\]

Current Increase Fraction (\( CIF \)) of a task sequence is a measure of non-decreasing trends in the current discharge profile. The lower its value, the less number of increasing current transitions are there in the discharge profile. We define \( CIF \) as follows where \( c \) denotes the chosen design point for any task \( k \):
4.1 Algorithm Description

During each iteration of the algorithm a valid schedule is created and a design-point assignment is chosen. In subsequent iterations the solution is improved such that the amount of battery capacity usage is less than the previous iteration. The algorithm terminates once the solution is not improved during two consecutive iterations. We start choosing design-points from a last design-point work our way up to the first task. During the selection of the design-point for any particular task the suitability of the each design-point \( B \) is calculated. The design-point which has the lowest value of \( B \) is chosen and the task is fixed to that design-point. Each task can be in three different states of design-point allocation: free, tagged and fixed. A task is tagged when we are evaluating the suitability of one of its design-points. Tasks which are neither tagged nor fixed are called free tasks. We use an heuristic approach for searching the design-space which involves the use of a window function (Explained later) until all design-points are considered. Once all tasks are assigned to design-points the battery capacity used is calculated and we sequence the tasks according to a weight assignment which is based on the current consumption of the design-points assigned to different tasks in this iteration (Explained in Section 4.). This new sequence is then used for the next iteration.

BatteryAwareSQNDPAlocation is the top level algorithm for task sequencing and design-point assignment and is given in Figure 1. \( \text{MinBCost} \) contains the minimum battery cost for any given iteration and is initialized to infinity. \( \text{PrevIterCost} \) contains the cost of the previous iteration for comparison pur-

**BatteryAwareSQNDPAlocation**

**Begin**

\[ \text{MinBCost} = \infty, \text{PrevIterCost} = \infty \]

\[ \text{L=SequenceDecEnergy(D,J)} \]

\[ \text{success} = \text{TRUE} \]

**while** (success)

\[ (\{\text{MinBCost},S\} = \text{EvaluateWindows(L,E,I,D)} \]

\[ \text{Ltemp} = \text{FindWeightedSequence}(L,I,D) \]

\[ \text{TempCost} = \text{CalculateBatteryCost}(Ltemp,S,I,D) \]

**if** TempCost < MinBCost **then**

\[ \text{MinBCost} = \text{TempCost} \]

\[ \text{success} = \text{TRUE} \]

**end if**

**if** MinBCost >\= PrevIterCost **then**

\[ \text{success} = \text{FALSE} \]

**else**

\[ \text{PrevIterCost} = \text{MinBCost} \]

**end if**

\[ \text{L} = \text{Ltemp} \]

**end while**

\[ (\{\text{MinBCost},S\} = \text{EvaluateWindows(L,E,I,D)} \]

\[ \text{Wflag} = \text{TRUE} \]

\[ \text{WindowStart} = m-1 \]

**while** (Wflag)

\[ \text{if} (d < CT(\text{WindowStart})) \text{ then} \]

\[ \text{WindowStart} = \text{WindowStart} - 1 \]

\[ \text{Wflag} = \text{TRUE} \]

**if** (WindowStart = 1) AND (d < CT(\text{WindowStart})) **then**

\[ \text{Exit with error // The deadline cannot be met} \]

**end if**

**else**

\[ \text{Wflag} = \text{FALSE} \]

**end if**

**end while**

\[ \text{while} \ \ (\text{windowstart} >= 1 ) \]

\[ //\text{Use WindowStart ... m columns of S, I, D for this iteration} \]

\[ \text{Stemp} = \text{ChooseDesignPoints}(L,E,I,D,\text{WindowStart},d) \]

\[ \text{TempCost} = \text{CalculateBatteryCost}(L,\text{Stemp},I,D) \]

**if** TempCost < MinBCost **then**

\[ \text{MinBCost} = \text{TempCost}; \]

\[ S = \text{Stemp} \]

**end if**

\[ \text{WindowStart} = \text{WindowStart} - 1 \]

**end while**

**return** (MinBCost, S)

\[ (S) = \text{ChooseDesignPoints}(L,E,I,D,\text{WindowStart},d) \]

**Initialize** S, Free all tasks in E

\[ \text{Tsum} = \text{D}(n,m) \]

\[ \text{Tsum} = 0 \]

Mark the task \( n \) in E to be fixed

\[ \text{S}(n,m) = 1 \] //Assign nth task to the lowest power design-point

**for** \( i = n-1 \) **downto** 1 **do**

\[ \text{for} \ j = m \text{ downto } \text{WindowStart} \text{ do} \]

\[ \text{S}(i,j) = 1 \]

\[ \text{Tsum} = \text{Tsum} + \text{D}(i,j) \]

Set task \( i \) to be tagged in S and fix it in E

\[ \text{SR} = (d - \text{Tsum})/d \]

\[ \text{CR} = (\text{ENR}, \text{CIF}, \text{DPF}) = \text{CalculateDPF}(E,S,I,D,\text{WindowStart},i,d) \]

\[ \text{B}(i,j) = \text{SR} + \text{CR} + \text{ENR} + \text{CIF} + \text{DPF} \]

\[ \text{S}(i,j) = 0, \text{Ttemp} = 0 \]

**end for**

**find** \( \text{B}(i,k) \) the minimum value of B for task i

\[ \text{S}(i,k) = 1 \]

Set task i to be fixed in S and E

\[ \text{Tsum} = \text{Tsum} + \text{D}(i,k) \]

**end for**

**return** S

**Figure 1. Algorithms for Battery-Aware Task Sequencing and Design-Point Allocation**
### Choosing Design Points

Choosing Design Points: In `ChooseDesignPoints()` we first initialize $S$ and set the state of all nodes in $E$ to be free. Recall that $E$ is the Energy Vector. We will use $E$ as a priority function while evaluating DPF. We start from the last task in the sequence and fix it to the lowest power design-point and move up towards the first task. $Tsum$ keeps track of the sum of the execution times of the tasks fixed so far. $Ttemp$ is used to keep track of the sum of execution time of the tagged tasks as well as the sum of the execution time of the fixed tasks. We evaluate the suitability $B$ of each design point $j$ of each task one by one. For any given task $i$ the design-point $k$ which has the minimum value of $B$ is chosen and the task $i$ is fixed to design point $k$ and the value of $Tsum$ is updated to reflect the chosen design-point for task $i$.

### Calculation of DPF, CIF and ENR

The algorithms given in Figure 2 are used to calculate DPF, CIF and ENR. Initially `CalculateDPF()` is called from `ChooseDesignPoints()` Then it calls `CalculateFactors()` to calculate ENR and CIF. Copies of $S$ and $E$ are made as $Stemp$ and $Ettemp$.

At any given point during the execution there will be some tasks in $S$ which have been fixed, tagged and free. Similarly, each task in $E(Stemp)$ has two states: fixed and free. A task is fixed in $E$ if it is fixed or tagged in $S$ and also when the highest powered design-point is chosen for it. If the deadline is not being met by choosing the lowest power consuming design-points of all free tasks in $E(Stemp)$, moving the first free task in $E(Stemp)$ from lower to a higher power consuming design-point decreases the execution time in the hope that the deadline will be met with the least increase in the overall energy consumption. For example, consider the tasks and their design-points shown in Figure 4. Here tasks $T5$ and $T4$ have been fixed to $DP4$ and $DP1$, respectively. $T3$ is the tagged task and we are calculating the DPF of $DP2$. Figure 4-a shows the initial condition at the start of calculation of DPF. Notice tasks $T1$ and $T2$ are both initially assigned to $DP4$. Suppose, that it is found that this assignment does not meet the deadline. Now we wish to use a higher powered DP such that the deadline is met. We look at $Eogency$ vector $E$ and find that tasks $3, 4, 5$ are fixed. The first free task in $E$ is $T1$. Therefore, it is assigned the next higher powered design-point, $DP3$ (Figure 4-b). Suppose, that this assignment also, does not meet the deadline. The assignment of $T1$ is moved up to $DP2$ (Figure 4-c). Now, let us assume that the assignment meets the deadline. Therefore, no further
moves are necessary. From Equation 2 and Equation 3, \( f = 1/3 \) and the number of free nodes \( x \) is 2. \( F_1 = 1/2, F_2 = 0, F_3 = 1/2, F_4 = 0, \) therefore \( DPF = 1/3 \) for task T3, DP2.

Despite the unusually large values for the task durations used in this example, the algorithm is equally applicable to any chosen time scale. Table 2 shows the task sequences generated for the four iterations of the algorithm along with the design-points (DP) assigned for each sequence. A “w” after a sequence number indicates the weighted sequence calculated for the corresponding iteration.

**Table 1. Data for example task graph G3**

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Design Point 1</th>
<th>Design Point 2</th>
<th>Design Point 3</th>
<th>Design Point 4</th>
<th>Design Point 5</th>
<th>Parents</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>917</td>
<td>7.3</td>
<td>563</td>
<td>11.2</td>
<td>288</td>
<td>15.0</td>
</tr>
<tr>
<td>T2</td>
<td>519</td>
<td>11.2</td>
<td>319</td>
<td>17.3</td>
<td>163</td>
<td>23.1</td>
</tr>
<tr>
<td>T3</td>
<td>611</td>
<td>5.9</td>
<td>375</td>
<td>9.2</td>
<td>192</td>
<td>12.2</td>
</tr>
<tr>
<td>T4</td>
<td>938</td>
<td>5.3</td>
<td>576</td>
<td>8.2</td>
<td>295</td>
<td>10.9</td>
</tr>
<tr>
<td>T5</td>
<td>781</td>
<td>4.0</td>
<td>480</td>
<td>6.1</td>
<td>246</td>
<td>8.2</td>
</tr>
<tr>
<td>T6</td>
<td>800</td>
<td>4.6</td>
<td>491</td>
<td>7.1</td>
<td>252</td>
<td>9.5</td>
</tr>
<tr>
<td>T7</td>
<td>720</td>
<td>7.3</td>
<td>442</td>
<td>11.2</td>
<td>126</td>
<td>15.0</td>
</tr>
<tr>
<td>T8</td>
<td>600</td>
<td>5.3</td>
<td>368</td>
<td>8.2</td>
<td>189</td>
<td>10.9</td>
</tr>
<tr>
<td>T9</td>
<td>650</td>
<td>4.6</td>
<td>399</td>
<td>7.1</td>
<td>204</td>
<td>9.5</td>
</tr>
<tr>
<td>T10</td>
<td>710</td>
<td>5.9</td>
<td>436</td>
<td>9.2</td>
<td>223</td>
<td>12.2</td>
</tr>
<tr>
<td>T11</td>
<td>500</td>
<td>6.6</td>
<td>307</td>
<td>10.2</td>
<td>157</td>
<td>13.6</td>
</tr>
<tr>
<td>T12</td>
<td>510</td>
<td>4.6</td>
<td>313</td>
<td>7.1</td>
<td>160</td>
<td>9.5</td>
</tr>
<tr>
<td>T13</td>
<td>700</td>
<td>4.0</td>
<td>430</td>
<td>6.1</td>
<td>226</td>
<td>15.0</td>
</tr>
<tr>
<td>T14</td>
<td>400</td>
<td>5.3</td>
<td>246</td>
<td>8.2</td>
<td>126</td>
<td>10.9</td>
</tr>
<tr>
<td>T15</td>
<td>380</td>
<td>3.3</td>
<td>233</td>
<td>5.1</td>
<td>119</td>
<td>6.8</td>
</tr>
</tbody>
</table>

**Table 2. : Task Sequences of G3 for different iterations**

<table>
<thead>
<tr>
<th>Iter</th>
<th>Seq No</th>
<th>Task Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>T1,T4,T5,T7,T3,T2,T6,T8,T10,T9,T13,T11,T12,T14,T15</td>
</tr>
<tr>
<td></td>
<td>S1w</td>
<td>T1,T3,T2,T4,T5,T6,T7,T8,T10,T9,T13,T12,T11,T14,T15</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>T1,T3,T2,T4,T5,T6,T7,T8,T10,T9,T13,T12,T11,T14,T15</td>
</tr>
<tr>
<td></td>
<td>S2w</td>
<td>T1,T3,T2,T4,T5,T6,T7,T8,T10,T9,T13,T12,T11,T14,T15</td>
</tr>
<tr>
<td>3</td>
<td>S3</td>
<td>T1,T3,T2,T4,T5,T6,T7,T8,T10,T9,T13,T12,T11,T14,T15</td>
</tr>
<tr>
<td></td>
<td>S3w</td>
<td>T1,T3,T2,T4,T5,T6,T7,T8,T10,T9,T13,T12,T11,T14,T15</td>
</tr>
<tr>
<td>4</td>
<td>S4</td>
<td>T1,T2,T4,T5,T7,T3,T6,T8,T10,T9,T13,T12,T11,T14,T15</td>
</tr>
<tr>
<td></td>
<td>S4w</td>
<td>T1,T2,T4,T5,T7,T3,T6,T8,T10,T9,T13,T11,T12,T14,T15</td>
</tr>
</tbody>
</table>

Table 3 gives the values of battery capacity used \( \sigma \) given in milli-Ampere-minutes along with the duration of the task sequence \( \Delta \) given in minutes, for each sequence and its weighted counterpart for the four iterations of the algorithm execution. The column marked “Win 1:5” contains the results for the battery capacity and the execution time when the algorithm was allowed to consider all five design points for all tasks. The data for each subsequent window where the number of allowed design points for consideration were decreased by one is also given in Table 3. The column called “Min \( \sigma \)” contains the minimum value of battery capacity chosen from
among the four different windows evaluated during an iteration. The last column gives the execution time corresponding to the window chosen which uses the minimum battery capacity. The battery capacity after the first iteration is 16353 mA min which decreases after each subsequent iteration until iteration 3. For iteration 4 there is no improvement in the value of the battery capacity used and the algorithm terminates. Also notice that for each iteration a valid schedule is generated which satisfies the deadline.

Table 3: Algorithm execution data for different iterations for G3

<table>
<thead>
<tr>
<th>Seq No</th>
<th>Win 1:5</th>
<th>Win 2:5</th>
<th>Win 3:5</th>
<th>Win 4:5</th>
<th>MinG</th>
<th>( \Delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>17169</td>
<td>229.8</td>
<td>17837</td>
<td>228.4</td>
<td>17038</td>
<td>227.1</td>
</tr>
<tr>
<td>S1w</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
</tr>
<tr>
<td>S2</td>
<td>14725</td>
<td>229.2</td>
<td>16126</td>
<td>229.2</td>
<td>15929</td>
<td>229</td>
</tr>
<tr>
<td>S2w</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
</tr>
<tr>
<td>S3</td>
<td>13737</td>
<td>229.8</td>
<td>16033</td>
<td>229.2</td>
<td>16061</td>
<td>229.8</td>
</tr>
<tr>
<td>S3w</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
</tr>
<tr>
<td>S4</td>
<td>13737</td>
<td>229.8</td>
<td>15866</td>
<td>229.3</td>
<td>16240</td>
<td>229.2</td>
</tr>
<tr>
<td>S4w</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
<td>- - - - - - -</td>
</tr>
</tbody>
</table>

5. Case Study

We present a real-world application of a robotic arm controller implemented on a voltage scalable processor as described in [10][1] to demonstrate the usefulness of this algorithm. The task-graph of this application (called G2) is shown in Figure 5 along with its various design-points. For G2 the task durations were proportional to the worst case execution of the tasks and were made inversely proportional to the scaling factor with respect to voltage of DP4 (V4) and task currents of the tasks and were made inversely proportional to the scaling factor with respect to V4. The battery capacities used for the four design-points with respect to V4 were as follows: 2.5, 1.66, 1.25, 1. The battery capacities used for three different deadlines (55, 75, 95 minutes) are shown in Table 3.

![Figure 5. Task Graph G2 and Design-Point Data](image)

Comparison with an Approach in [1]: We compared the results from our algorithm to a method in [1] where the design points were chosen using a dynamic program such that the total energy used is minimized and a given deadline is met. In the algorithm given in [1], after the design-point allocation a greedy sequencing of all tasks in the task graph \( G(V,E) \) was performed where the tasks were assigned a weight according to Equation 5, where \( G_v \) is the subgraph rooted at node v and \( \text{MeanI}(G_v) \) is the mean current of all nodes in the subgraph rooted at node v. Whenever a node is to be scheduled the node with the largest weight was selected among the nodes in the ready list.

\[ \forall v \in G \ w(v) = \max \{ I_v, \text{MeanI}(G_v) \} \quad (5) \]

We executed the two algorithms for three different deadlines for the two different test task graphs G2 and G3 discussed earlier. We present the data for this comparison in Table 4. Notice that as the deadline increases the amount of battery capacity used decreases. This is because the algorithm can choose design points which have lower performance but also consume less capacity. Also, we see that our algorithm gives better results for the two task graphs under consideration.

Table 4: Comparison of our algorithm with an approach in [1]

<table>
<thead>
<tr>
<th>Deadline (minutes)</th>
<th>G2: 9 Nodes, 4DPs</th>
<th>G3: 15 Nodes, 5 DPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>75</td>
<td>95</td>
</tr>
<tr>
<td>Batt. Capacity by Our Algo (mA)</td>
<td>30913</td>
<td>13751</td>
</tr>
<tr>
<td>Batt. Capacity by Algo [1] (mA)</td>
<td>35739</td>
<td>13885</td>
</tr>
<tr>
<td>% Diff</td>
<td>13.6</td>
<td>0.9</td>
</tr>
</tbody>
</table>

References


