# A Constraint-based Application Model and Scheduling Techniques for Power-aware Systems *

Jinfeng Liu, Pai H. Chou, Nader Bagherzadeh, Fadi Kurdahi
Dept. of Electrical & Computer Engineering
University of California at Irvine
Irvine, CA 92697-2625 USA

{jinfengl, chou, nader, kurdahi}@ece.uci.edu

## ABSTRACT

New embedded systems must be *power-aware*, not just low-power. That is, they must track their power sources and the changing power and performance constraints imposed by the environment. Moreover, they must fully explore and integrate many novel power management techniques. Unfortunately, these techniques are often incompatible with each other due to overspecialized formulations or they fail to consider system-wide issues. This paper proposes a new graph-based model to integrate novel power management techniques and facilitate design-space exploration of power-aware embedded systems. It captures min/max timing and min/max power constraints on computation and non-computation tasks through a new constraint classification and enables derivation of flexible system-level schedules. We demonstrate the effectiveness of this model with a power-aware scheduler on real mission-critical applications. Experimental results show that our automated techniques can improve performance and reduce energy cost simultaneously. The application model and scheduling tool presented in this paper form the basis of the IMPACCT system-level framework that will enable designers to aggressively explore many power-performance tradeoffs with confidence.

## Keywords

constraint modeling, power-aware real-time scheduling, embedded systems software, system-level design

## 1. INTRODUCTION

Power management is becoming a central issue in embedded systems. It is particularly critical to systems that must carry their own energy source. As these systems are deployed in an diverse range of operating conditions, they must be able to adapt to radically different constraints by shifting their power/performance curves.

An example of such a system is the NASA Mars Pathfinder developed at JPL [1]. It draws power from a battery pack and a solar panel to operate under extremely harsh weather and power conditions for hundreds of days. Such systems pose new challenges to hardware/software co-design in design space exploration. It is no longer sufficient to find just one good design point; instead, the same design must cover a range of behaviors depending on the operating conditions.

In this section, we sketch our motivating example, and highlight several key issues: (1) the need for *power-aware* designs, rather than just *low-power*; (2) the need for power management for the *entire system* that includes thermal and mechanical subsystems, not just the digital computer or microprocessors; (3) the need to integrate novel power management techniques into the same co-design framework. To address these issues, we propose a new system-level model that captures the essential power/performance features in a concise form. It enables our constraint-driven power manager to meet both min/max timing constraints and unsteady min/max power constraints for the entire system.

### 1.1 Example: the Mars Pathfinder

The NASA/JPL Mars Pathfinder is a rover designed to roam on the surface of Mars to several target locations for an extended period of time. Its tasks include performing experiments and taking snapshots and transmitting them wirelessly back to the spaceship. Since the temperature on Mars can be as low as $-80\,^{\circ}\mathrm{C}$, the motors must be kept heated periodically in order to turn normally. It also uses a laser-guided hazard detection system for its navigation. During daytime, the rover draws most of its power from the solar panel for its operations. Any excess power cannot be stored, since the battery pack is not rechargeable (for weight/efficiency concerns); instead, it is dumped into the heater for natural dissipation. At night, the system draws power from the battery pack to remain in sleep mode and to keep it warm above a minimum temperature.

The designers of the rover were extremely concerned about power usage and took a conservative approach: they serialized all power draws. This allows the rover to operate with very low battery power indeed, which enables it to run for hundreds of sols (Martian days) during daylight. However, full serialization also means the rover moves as slowly as 10cm per minute, and it can take a total of only three pictures per day.

### 1.2 Power-aware vs. low-power

Traditionally, many components and systems have been designed to be *low-power*. The existing Mars rover is an example of a low-power design, whose goal is to minimize power usage. Another mission to Mars or other planets may have a different set of objectives, e.g., shorter mission completion time, taking more pictures, or traveling a farther distance. In these cases, the low-power design is too rigid and cannot support all these power/performance trade-

*This research was sponsored by DARPA under contract F33615-00-1-1719

offs. Instead, it must be designed to be *power-aware*, which means making the best use of its available power. Power-aware designs subsume low-power as a special case: it must handle not only max-power constraints (for low power cases), but min-power constraints as well.

A power-aware design can greatly improve the utility of the rover. We observe that the battery is non-rechargeable, and thus excess solar power would be wasted. In the existing design, the rover follows the same serial schedule regardless of the solar power level. A rover with more parallelism in its schedule can perform better while saving even more battery energy than the existing low-power design if it can take advantage of the free solar power, as validated by our experimental results. Our goal is to make this power/performance "knob" available at runtime so the power manager can tune itself as needed to match the changing conditions.

## 1.3 System-level power-aware design

Power-aware designs must be done at the system-level, not just at the component level. Amdahl's law applies to power as well as performance. That is, the power saving of a given component must be scaled by its percentage contribution to the entire system. Therefore it is critical to identify where power is being consumed in the context of a system, not just isolated components in the digital subsystem. Instead, all relevant power consumers must be included. In the case of the Mars rover, it turns out that the major power consumers are the motors and the heaters rather than the digital computer. A successful power-aware design must consider these non-computation domains and coordinate their power usage as a whole system. Also, power supplies with different characteristics must be accounted for while making power management decisions.

## 1.4 Related work

Prior works have addressed minimization of power usage while maintaining a satisfactory level of performance or meeting real-time constraints. However, these low-power techniques often cannot be directly adapted in power-aware systems due to limited compatibility between application models.

Shutting down idle subsystems can save a significant amount of power in a system. Proposed improvements either attempt to make the timeout adaptive to the actual usage pattern, or predict the proper time to shutdown and power up subsystems [6, 3, 7]. Unfortunately, these techniques have several limitations. First, they do not handle timing constraints, including deadlines and min/max separation. Second, they are not power-aware in the sense that they do not distinguish between free power (e.g., solar sources) vs. expensive power (non-rechargeable battery). Rate-monotonic scheduling has been extended to scheduling variable-voltage processors. The idea is to slow down the processor just enough to meet the deadlines [5, 4]. Such techniques also have limitations. First, they are CPU schedulers that minimize CPU power, rather than power managers that control subsystems and task executions. Second, in practice, it may be difficult to tune the voltage or frequency scale of processors. As a result, the risk of missing deadlines may be high, even if the context switching overhead is taken into account. While these schedulers meet deadline-based timing constraints, they do not handle power usage as hard constraints.

## 1.5 Application models and design tools

Our approach is to support power-aware designs with a system-level design tool built on an application model that sufficiently captures task characteristics of the application. One lesson learned from the rover was that without such tools, designers have no choice but to hand-craft many power management decisions in the implementation, resulting in a conservative design that could not consider more alternatives. The purpose of our tool is to enable the exploration of many more points in the design space, so that additional knowledge about the mission can be incorporated to refine the design without requiring dramatic redesigns. The basis of our tool-based methodology is to capture different timing relationships and power characteristics in a constraint-based model that effectively exposes the potential design alternatives to assist power management decisions.

The work presented in this paper represents one of the core tools in a larger co-design framework, called IMPACCT. The designer inputs high-level behavioral specifications of the design in terms of communicating tasks and constraints. The specifications are reconstructed into a constraint graph based on the application model to assist the power-aware scheduler. The output is then fed to another tool that performs optimizations and synthesis of power managers at the architectural level.

This paper is organized as follows. We present our application model in Section 2 and graph-based scheduling algorithms in Section 3. We discuss experimental results in Section 4 and highlight several design points exposed by our model, which were otherwise overlooked in the existing hand-crafted design.

# 2. PROBLEM FORMULATION

Our problem formulation is based on an extension to a constraint graph used in a time-driven scheduling problem [2]. One distinction we make in this paper is that we represent one task by its start event and end event. This separation enables a new constraint classification that exposes some key characteristics of this constraint graph representation.

## 2.1 Problem definition

We formally define the objects in our model, including tasks, constraints, resources, operational modes, and schedules.

**Definition 1 (Task $x \in T$)** A task $x$ is characterized by a set representation in that $x = \{s_x, e_x, \tau_x, \chi_x, M_x, m_x, d_x, p_x\}$, where $s_x$ and $e_x$ are the start and end events, respectively; $\tau_x$ is a generalized workload of the task; $\chi_x$ is the power characteristic of the task; $M_x$ is the execution resource to which the task is mapped; $m_x \in M_x$ is the selected operational mode of resource $M_x$; $d_x$ is the execution delay; $p_x$ is the power profile function of the task over time in its execution duration.

Among these attributes, $s_x$, $e_x$, $\tau_x$ and $\chi_x$ are intrinsic to the task itself and will remain immutable over partitions and mode switches. We use a pair of events $s_x$, $e_x$ to capture the timing requirements of the task itself, e.g., the variable execution delay on different mode selections. Such timing relationships are partially related to workload $\tau_x$, in that execution time $d_x$ depends on $\tau_x$. Power characteristic $\chi_x$ is a general description of power behavior, e.g., constant, linear, exponential, etc. It must be specified explicitly before the power profile $p_x$ is given based on mode selection $m_x$. This power property can be adapted to various forms that either characterize different subsystems or are application-specific. The other attributes $M_x$, $m_x$, $d_x$, and $p_x$ are determined by the scheduler. Resource mapping $M_x$ is normally predefined in allocation. The scheduler can also override it if migration is allowed. Mode selection $m_x$ can be arranged by the scheduler either statically or dynamically. Execution delay $d_x$ is a function of $\tau_x$, $M_x$ and $m_x$. $p_x$ is a function of time with parameters $d_x$, $M_x$ and $m_x$, it gives the power profile of task $x$ as $p_x(t), 0 \leq t \leq d_x$.

**Definition 2 (Resource $M$ with operational modes $m$)** An execution resource $M$ is a set of valid operational modes, $M = \{m_i\}, i = 1, \ldots, n$. An operational mode $m$ is a collection $m = (\alpha_m, f_m)$, where its characteristic parameters are included in $\alpha_m$, $f_m = (d_m, p_m)$ is a series of functions including delay function $d_m(x)$ and power function $p_m(x)$ to describe the behaviors of a task $x$ that is executed in this mode.

The parameter set $\alpha_m$ consists of attributes of mode $m$ such as voltage, clock rate, bandwidth, etc. The function set $f_m$ includes several functions that characterize tasks executed in this mode. For a task $x$ executed in mode $m$, the delay function $d_m(x)$ calculates the execution delay $d_x$ of task $x$ based on mode parameters $\alpha_m$ and workload $\tau_x$; the power function $p_m(x)$ gives task $x$'s power profile $p_x(t)$, which is a function of mode parameters $\alpha_m$, delay $d_x$, and power characteristic $\chi_x$.

**Definition 3 (Timing constraints)** A timing constraint specifies the timing relationship between two events $u$ and $v$, in one of the two forms:
(1) A *min timing constraint* $u \rightarrow v : \delta, \delta \geq 0$ indicates that $v$ must happen at least $\delta$ time units after $u$ happens, formally $t_v - t_u \geq \delta$.
(2) A *max timing constraint* $u \leftarrow v : \delta, \delta > 0$ indicates that $v$ must happen at most $\delta$ time units after $u$ happens, formally $t_v - t_u \leq \delta$.

This specification handles general timing relationships between events. A deadline is a special case of a max timing constraint from a task's end event to the *anchor*, the start-event for the schedule.

**Definition 4 (Schedule $\sigma$)** Given a task set $T$ and a resource set $R$, a schedule $\sigma$ maps a task $x \in T$ to an operational mode $m_x \in M_x \in R$, and assigns start and end times $t_{s_x}, t_{e_x}$ to events $s_x$ and $e_x$. Without ambiguity, we further overload the $\sigma$ notation to map any event $u$ to its assigned time according to $\sigma$, that is, $t_u = \sigma(u)$.

The time assignment to $s_x$ and $e_x$ must be consistent with the execution delay $d_x$, $\sigma(e_x) - \sigma(s_x) = d_x$. The power profile of a schedule $\sigma$ can be accumulated from the power profiles of each task, $p_\sigma(t) = \sum p_x(t - \sigma(s_x))$, $\forall$ task $x \in T$ and time $t$ such that $\sigma(s_x) \leq t \leq \sigma(e_x)$.

## 2.2 Constraint graph and its properties

**Definition 5 (Constraint graph $G(V,E)$)** Given a task set $T$, a timing constraint set $C$, a constraint graph $G(V,E)$ can be constructed as follows. The vertices $V$ represent events $\{anchor\} \cup \{s_x, e_x\}$, $\forall x \in T$, where *anchor* represents the virtual start-event that precedes all other events. The edges $E \subseteq V \times V$ represent timing relationships between events. For two vertices $u, v \in V$, an edge $(u,v)$ with weight $w(u,v)$ is denoted as $(u,v) : w(u,v)$. It specifies the timing relationships of event $u$ and $v$, such that $t_v - t_u \geq w(u,v)$.

Three types of edges represent three different types of timing relationships between two events. They are defined as follows.

**Definition 6 (Constraint edges $(u,v) : \delta$)** Each timing constraint in $C$ is represented by a constraint edge in the constraint graph $G$.
(1) A min timing constraint $u \rightarrow v : \delta$ is represented by an edge $(u,v) : \delta$ with weight $\delta \geq 0$.
(2) A max timing constraint $u \leftarrow v : \delta$ is represented by an edge $(v,u) : -\delta$ with weight $-\delta < 0$.

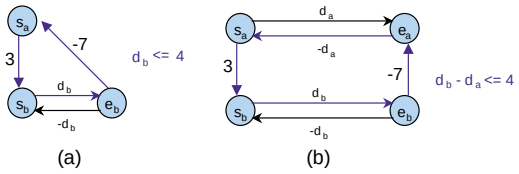


**Figure 1: Extra constraints extracted by preprocessing**

**Definition 7 (Duration edges $(s_x,e_x) : d_x$ and $(e_x,s_x) : -d_x$)** The execution delay $d_x \geq 0$ of a task $x \in T$ is represented by a pair of duration edges in the constraint graph $G$: edge $(s_x,e_x) : d_x$, and edge $(e_x,s_x) : -d_x$.

**Definition 8 (Serialization edges $(e_x,s_y) : 0$)** A serialization edge $(e_x,s_y) : 0$ is an added edge to serialize task $x$ before $y$.

Among all three types of edges, constraint edges represent timing constraints between tasks and always remain constant. The weights on duration edges, which represent execution delays of tasks, can be changed according to mode selection. Serialization edges can be added and removed by the scheduler. Tasks that share the same resource must be serialized to prevent resource conflict. The scheduler can also serialize tasks to avoid exceeding maximum supply power.

**Lemma 1 (Schedulability properties)** Given a scheduling problem formulated as a constraint graph, the time assignments by a schedule $\sigma$ can be computed as the single source longest path lengths from the anchor vertex on the constraint graph. A positive cycle in the graph indicates a conflicting set of timing requirements that cannot be satisfied.

**Corollary 1 (Extension to schedulability properties)**
(1) If a positive cycle consists of only constraint edges, the problem is not schedulable.
(2) If a positive cycle contains duration edges, the problem may be solved by changing the operational modes of corresponding tasks.
(3) If a positive cycle contains serialization edges, the problem may be solved by altering the partial ordering to serialized tasks.

Lemma 1 and Corollary 1 can be used to discover design points that are implied in the problem. Fig. 1 shows an example of how some extra constraints can be extracted by preprocessing the graph. Since the delays are functions of modes, the inequalities involving delays can be viewed as rules to mode selections on related tasks.

A valid schedule can be defined based on the constraint graph and its properties.

**Definition 9 (Validity of a schedule)** Given a constraint graph $G$ constructed from a task set $T$ and a constraint set $C$, and a resource set $R$ to which all tasks in $T$ are mapped, a schedule $\sigma$ is valid if
(1) $G$ is schedulable by Lemma 1, and
(2) $\forall$ tasks $x, y \in T$ such that $M_x = M_y \in R$, $x$ and $y$ must be serialized, that is, either $\sigma(e_x) \leq \sigma(s_y)$ or $\sigma(e_y) \leq \sigma(s_x)$ holds.

## 2.3 Slack properties of a valid schedule

Given a valid schedule, slack is a measure of how much an event can be safely rearranged to another time without invalidating the schedule.

**Definition 10 (Constraint edge slack)** For a given valid schedule $\sigma$, $\forall$ constraint edges $(u,v):\delta$ in the constraint graph $G$, the slack function of edge $(u,v)$ is defined as $slack(\sigma,u,v) = \sigma(v) - \sigma(u) - \delta$.

The slack value must be non-negative, otherwise the schedule is not valid. The constraint edge slack exposes the bounds on re-assigning time slots to events $u$ and $v$ (by delaying $u$ or executing $v$ earlier) without violating timing constraint represented by edge $(u,v):\delta$. We do not summarize the slack properties for serialization edges and duration edges. This is because serialization edges are not necessary constraints for schedulability; and the slacks of duration edges are zero by definition.

**Definition 11 (Constraint slack interval $\Delta_t$)** For a given schedule $\sigma$ for a constraint graph $G$,
(1) The *forward constraint slack* of an event $u$ is the minimum among all edge slacks of $u$'s outgoing constraint edges,
$forward\_slack(\sigma,u) = min(slack(\sigma,u,v))$, $\forall$ constraint edges $(u,v)$.
(2) The *backward constraint slack* of an events $u$ is the minimum among all edge slacks of $u$'s incoming constraint edges,
$backward\_slack(\sigma,u) = min(slack(\sigma,v,u))$, $\forall$ constraint edges $(v,u)$.
(3) The *constraint slack interval* $\Delta_t(\sigma,u)$ of an event $u$ is the interval $[\sigma(u) - backward\_slack(\sigma,u), \sigma(u) + forward\_slack(\sigma,u)]$ in the time dimension.

**Lemma 2** If a schedule $\sigma$ is valid, then a modified schedule $\sigma'$ does not violate any timing constraint if it is identical to $\sigma$ except $\sigma(u) \neq \sigma'(u)$, and $\sigma'(u) \in \Delta_t(\sigma,u)$, for a specific event $u$.

Lemma 2 exposes the available time space for an alternative time assignment to an event. However, such adjustment must not result in any resource conflict. Definition 12 indicates the vacant time slots to schedule a task without resource conflicts. The start and end events of a task can have different constraint slack intervals, while their resource slack intervals are defined to be identical.

**Definition 12 (Resource slots $\lambda_i$, resource slack intervals $\Delta_r$)** Given a valid schedule $\sigma$ and a task $x$,
(1) the $i^{th}$ *resource slot* of $x$ $\lambda_i(\sigma,x)$ is a closed interval in time dimension during which resource $M_x$ is not occupied by any tasks other than $x$.
(2) the *resource slack intervals* $\Delta_r(\sigma,x)$ of $x$ are a collection of all resource slots of $x$, $\Delta_r(\sigma,x) = \bigcup_i \lambda_i(\sigma,x)$.
(3) Events $s_x$ and $e_x$ have identical resource slots and resource slack intervals as those of task $x$, $\lambda_i(\sigma,s_x) = \lambda_i(\sigma,e_x) = \lambda_i(\sigma,x)$, $\Delta_r(\sigma,s_x) = \Delta_r(\sigma,e_x) = \Delta_r(\sigma,x)$.

**Definition 13 (Overall slack intervals $\Delta$)** Given a valid schedule $\sigma$ and an event $u$, the overall slack intervals $\Delta(\sigma,u)$ of $u$ are defined as $\Delta_t(\sigma,u) \cap \Delta_r(\sigma,u)$.

**Lemma 3** Given a valid schedule $\sigma$, a modified schedule $\sigma'$ neither violates any timing constraint nor causes any resource conflict, if it is identical to $\sigma$ except $\sigma(u) \neq \sigma'(u)$, and $\sigma'(u) \in \Delta(\sigma,u)$, for a specific event $u$.

**Lemma 4 (Slack-bounded schedulability)** Given a valid schedule $\sigma$, an alternative schedule $\sigma'$ to a task $x$ is valid if and only if:
(1) $\sigma'(s_x) \in \Delta(\sigma,s_x)$ and $\sigma'(e_x) \in \Delta(\sigma,e_x)$, and
(2) $\exists i$ such that $\sigma'(s_x) \in \lambda_i(\sigma,x)$ and $\sigma(e_x) \in \lambda_i(\sigma,x)$, and
(3) $\sigma'(e_x) - \sigma'(s_x) = d_x$, which is a function of mode $m_x$ on resource $M_x$ to execute task $x$.

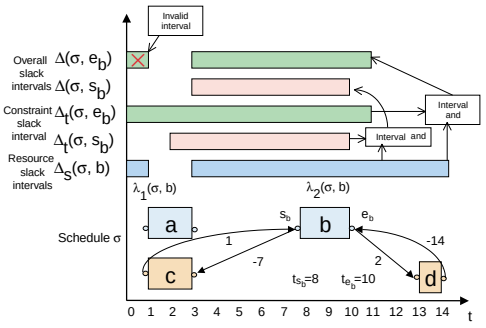


**Figure 2: Slack intervals of task $b$**

Lemma 4 exposes the space to all alternative schedules of a task in the time dimension. Such properties are especially meaningful to incorporate different power management decisions. Fig. 2 exemplifies the slack intervals of task $b$. Lemma 4 eliminates illegal combinations of $s_b$ and $e_b$.

## 2.4 Power consumption and power constraints

The power profile of a task can be calculated by the power function associated with the selected operational mode. In practice, the power consumption could be a range or in the form of (min, max, typical) or an exact number. These cases can be formulated as special cases of power functions. The global power profile of a schedule is the accumulation of power profiles of all tasks that are mapped into different time slots. This is constrained by two input parameters, max power and min power. The *max power* is a hard constraint: at any given moment, the total power consumption by all running tasks must not exceed this limit. The *min power* is a soft constraint: the scheduler should make the best effort to meet the min power goal. This will control the amount of jitter in power draw, as well as ensuring full utilization of free power such as solar.

# 3. SCHEDULING ALGORITHMS

Based on the constraint graph formulation, we develop graph algorithms for power-aware scheduling, shown in Fig. 3. We present an incremental approach to exploring a schedule that satisfies all constraints in three phases: (1) timing, (2) max power, (3) min power. First, we apply basic graph algorithms to solve all timing constraints. Second, starting with a valid schedule computed by step one, if max power constraint is violated at some time, we partially serialize tasks to decrease power consumption based on slack analysis. Finally, given a schedule that meets both timing and max power constraints, we reorder tasks within their slack intervals to match the min power constraint.

In the first phase, the scheduler performs a topological traversal on the graph by interleaving a start event with its corresponding end event so that one task is scheduled by visiting every two vertices. Serialization edges are added for tasks that share the same resource. This algorithm is proved to be able to find a valid schedule if one exists. It extends a previous serialization algorithm [2] to supporting parallel tasks on multiple execution resources.

In the second phase, the algorithm scans the schedule computed by the previous phase to find the time slots when the max power constraint is violated. The algorithm first attempts a partial reordering on simultaneous tasks to reduce power consumption. Several tasks are selected to be delayed within their slack intervals. If no tasks can be delayed, an arbitrary delay is enforced and a total re-

```
TimingConstraintScheduler(Graph G, vertex anchor, vertex c)
    La := SINGLE SOURCE LONGEST PATH(G, anchor)
    if (positive cycle found) then return FAIL
    C := set of topological successors of candidate c
    if (C = ∅) then return σ with σ(c) := La
    while (C ≠ ∅) do
        if (c is a start event) then v := end event of c
        else
            v := SelectSuccessor(C)
            C := C − {v}
            if (v is an end event) v := start event of v
B:          foreach u ∈ C do
                if u ∉ v's successors, then add u to v's successors
                if (M_c = M_u) then
                    u := start event of u
                    add serialization edge (c, u) to G
            w := the most recently scheduled end event, where (M_w = M_v)
            if (w ≠ nil) then add serialization edge (w, v) to G
        σ = TimingConstraintScheduler(G, anchor, v)
        if (σ ≠ FAIL) then return σ with σ(c) := La
        Undo changes to G since step B
        if (c is a start event) then return FAIL
    return FAIL
```

(a) Scheduling algorithm for timing constraints

```
MaxPowerConstraintScheduler(Graph G, vertex anchor, MaxPower)
    σ := TimingConstraintScheduler(G, anchor, anchor)
    if (σ = FAIL) then return FAIL
    for (t := 0; t ≤ execution time of σ; t := t + 1) do
        S := set of active tasks at t, ordered by forward slacks
        power := power consumption of all tasks in S
        reschedule := FALSE
        while (power > MaxPower or reschedule = TRUE) do
B:          repeat
                v := task with largest slack in S
                if (slack(v) = 0) then reschedule := TRUE
                delay v by some time units (heuristically determined)
                power := power − p(v)
                S := S − {v}
            until (power ≤ MaxPower or S = ∅)
            if (S = ∅) then return FAIL
            if (reschedule = TRUE) then
                lock start time of all tasks in S
                σ := MaxPowerConstraintScheduler(G, anchor, MaxPower)
                if (σ ≠ FAIL) then return σ
                Undo changes to G since step B
    return σ
```

(b) Scheduling algorithm for max power constraint

**Figure 3: Power-aware scheduling algorithms**

ordering to all tasks is performed. A valid schedule is found if the max power constraint is satisfied for all time slots in the schedule. The key issue in this algorithm is to properly select tasks to be delayed. We apply a slack-based heuristic ordering function so that the tasks with the largest slack will be selected first. Slack-based heuristics are appropriate in leading the algorithm to a feasible solution and converge more quickly.

In the third phase, the algorithm examines the valid schedule computed by the second phase and tries to reorder tasks within their slack intervals for min power. If such adjustments are not possible, the schedule is returned as a feasible solution in the sense that all hard constraints are satisfied. Since the algorithm can be easily developed based on slack analysis to the tasks, the illustration is omitted.

# 4. EXPERIMENTS AND RESULTS

We model the Mars rover in the proposed constraint graph, apply power-aware schedulers to the rover with changing power constraints, and compare the results with the schedules used in past missions. The timing constraints on the rover are summarized in Table 1, and the constraint graph is shown in Fig. 4. We investigate three cases where power constraints and power consumption vary with the environment: the best, typical and worst cases, with solar power levels at 14.9W, 12W, and 9W, respectively, as shown in Table 2. In all cases, the min power constraint to the rover is the available solar power. This solar power level plus 10W maximum battery power output constitutes the max power constraint.

We first examine the existing schedule shown in Fig. 5. JPL uses a completely serialized schedule that is low-power but not power-aware. This schedule is too conservative in cases where the available solar power is sufficient to support parallel operations. This can be revealed by examining the slack intervals of tasks. All heating tasks have large slack intervals, which indicate that serialization to these tasks is not necessary. Without a design tool, such an opportunity for improving performance and power utilization is usually overlooked.

Fig. 6 and 7 illustrate the schedules for the best case and typical case after applying power-aware scheduling. In the best case, we manually unroll the loop and insert two heating tasks to improve so-

| Operation | Duration(s) | Timing constraints |
|---|---|---|
| Heating steering motors | 5 | At least 5s, at most 50s before steering |
| Heating wheel motors | 5 | At least 5s, at most 50s before driving |
| Hazard detection | 10 | At least 10s before steering |
| Steering | 5 | At least 5s before driving |
| Driving | 10 | At least 10s before next hazard detection |

**Table 1: Timing constraints in Mars rover's operations**

| Tasks & Power sources | Duration(s) | Power(W) | | |
|---|---|---|---|---|
| | | -40 °C | -60 °C | -80 °C |
| Solar panel | | 14.9 | 12 | 9 |
| Battery | | 10 max | 10 max | 10 max |
| Heat one motor | 5 | 5.1 | 6.2 | 7.5 |
| Heat two motors | 5 | 7.6 | 9.5 | 11.3 |
| Drive | 10 | 7.5 | 10.9 | 13.8 |
| Steer | 5 | 4.3 | 6.2 | 8.1 |
| Hazard detection | 10 | 5.1 | 6.1 | 7.3 |
| CPU | Constant | 2.5 | 3.1 | 3.7 |

**Table 2: Power consumption of Mars rover's operations**

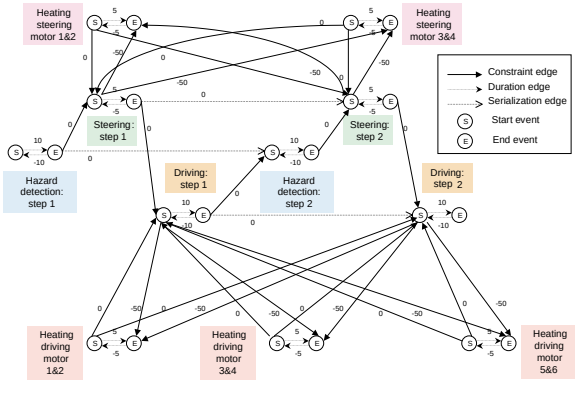


**Figure 4: Constraint graph for Mars rover's operations**

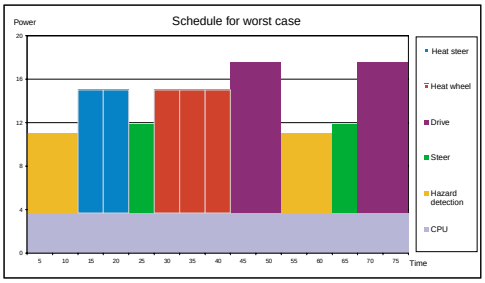


**Figure 5: The existing design only schedules for the worst case**

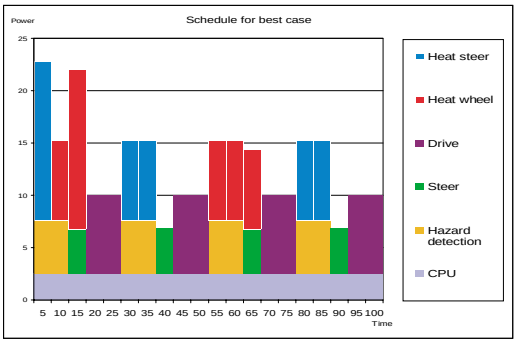


**Figure 6: Power-aware schedule for the best case**

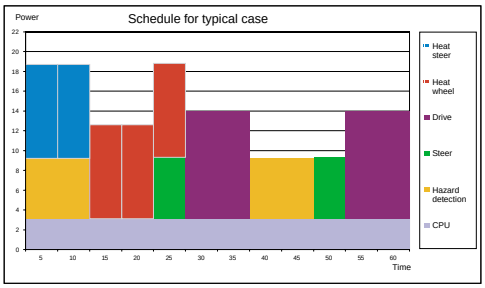


**Figure 7: Power-aware schedule for the typical case**

lar energy utilization. Since power budget is high, a fast schedule is given by allowing parallel tasks. In the typical case, parallel operations are still possible while some tasks are serialized. In the worst case, our scheduler produces the same fully serialized schedule as JPL's hand-crafted solution. The distinction is that our schedule is completely constraint-driven without hardwiring any decision into the implementaion. The schedules are compared in Table 3.

We use execution time and non-rechargeable energy cost as the metrics to evaluate the schedules. Although the rover under the existing scheme runs very slowly, such a low-power design draws less energy from the battery. To evaluate this power-performance trade-off, we apply the schedules to a mission scenario, where the available solar power varies over time. We assume the rover is traveling to a location that is 48 steps away. The mission starts from the best case. Then the solar power level drops from 14.9W to 12W after 10 minutes, and falls to the worst case at 9W 10 minutes later. Under the existing schedule, the rover will spend 10 minutes evenly in each of the three cases since its speed (16 steps per 10 minutes) does not track the available solar power. This results in a long execution time and high energy cost in the worst case for 10 minutes. When power-aware schedules are used, the rover finishes 50% of the work (24 steps) in the first 10 minutes, and 42% (20 steps) in

| Solar power (W) | Battery energy (J) | Solar energy(J) | Utilization of solar energy | Time(s) | Moving distance |
|---|---|---|---|---|---|
| 14.9 | 0 | 672.5 | 60% | 75 | 2 steps - 14cm |
| 12 | 55 | 817 | 91% | 75 | 2 steps - 14cm |
| 9 | 388 | 675 | 100% | 75 | 2 steps - 14cm |

(a) Performance of the rover under existing schedule

| Solar power (W) | Battery energy (J) | Solar energy(J) | Utilization of solar energy | Time(s) | Moving distance |
|---|---|---|---|---|---|
| 14.9 | 79.5(1st iter.) 6(rest) | 534 | 70% | 50 | 2 steps - 14cm |
| 12 | 147 | 679 | 94% | 60 | 2 steps - 14cm |
| 9 | 388 | 675 | 100% | 75 | 2 steps - 14cm |

(b) Performance of the rover under power-aware schedules

**Table 3: Performance-energy comparison of the two schedules**

| Time frame (s) | Solar power (W) | JPL | | | Power-aware | | |
|---|---|---|---|---|---|---|---|
| | | Travel distance | Time (s) | Energy cost (J) | Travel distance | Time (s) | Energy cost (J) |
| 0-599 | 14.9 | 16 | 600 | 0 | 24 | 600 | 145.5 |
| 600-1199 | 12 | 16 | 600 | 440 | 20 | 600 | 1470 |
| 1200 - | 9 | 16 | 600 | 3114 | 4 | 160 | 776 |
| Total | | 48 | 1800 | 3554 | 48 | 1360 | 2391.5 |
| | | | | | Improvement | 24.4% | 32.7% |

**Table 4: Comparison under a mission scenario**

the next 10 minutes, leaving remaining 8% (4 steps) executed in the worst case for less than 3 minutes. Therefore, the rover can finish the mission earlier before having to work in the costly worst case. The analysis in Table 4 shows power-aware schedules can win both on performance and energy savings considerably.

# 5. CONCLUSION

Successful power-aware designs require incorporation of the best power management techniques. A main obstacle that prevents their smooth integration is the hardwired assumptions in various formulations. In this paper, we propose a novel application model that represents the first step towards overcoming this inherent barrier. Our experiments on real applications demonstrate promising results by exposing non-obvious design points that yield both significant energy reduction and performance speedup at the system level. Such improvements are due to the synergy of several novel techniques, which would not be applicable separately.

# 6. REFERENCES


[1] NASA/JPL's Mars Pathfinder home page. http://mars3.jpl.nasa.gov/MPF/index0.html.

[2] P. Chou and G. Borriello. Software scheduling in the co-synthesis of reactive real-time systems. In *Proc. Design Automation Conference*, pages 1–4, June 1994.

[3] E.-Y. Chung, L. Benini, and G. De Micheli. Dynamic power management using adaptive learning tree. In *Proc. International Conference on Computer-Aided Design*, pages 274–279, 1999.

[4] I. Hong, M. Potkonjak, and M. B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proc. International Conference on Computer-Aided Design*, pages 653–656, November 1998.

[5] T. Okuma, T. Ishihara, and H. Yasuura. Real-time task scheduling for a variable voltage processor. In *Proc. International Symposium on System Synthesis*, pages 24–29, 1999.

[6] T. Simunic, L. Benini, and G. De Micheli. Event-driven power management of portable systems. In *Proc. International Symposium on System Synthesis*, pages 18–23, 1999.

[7] M. Srivastava, A. Chandrakasan, and R. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on VLSI Systems*, 4(1):42–55, March 1996.


# Acknowledgement


Special thanks to Dr. N. Aranki, Dr. B. Toomarian, Dr. M. Mojarradi and Dr. J. U. Patel from JPL for their assistance with the Mars rover application.