

# IMPACCT: Methodology and Tools for Power-Aware Embedded Systems

Pai H. Chou (chou@ece.uci.edu), Jinfeng Liu (jinfengl@ece.uci.edu),  
Dexin Li (dli@ece.uci.edu) and Nader Bagherzadeh  
(nader@ece.uci.edu)  
*Department of Electrical & Computer Engineering*  
*University of California, Irvine, CA 92697-2625 USA*

## Abstract.

Power-aware systems are those that must exploit a wide range of power/performance trade-offs in order to adapt to the power availability and application requirements. They require the integration of many novel power management techniques, ranging from voltage scaling to subsystem shutdown. However, those techniques do not always compose synergistically with each other; in fact, they can combine subtractively and often yield counterintuitive, and sometimes incorrect, results in the context of a complete system. This can become a serious problem as more of these power aware systems are being deployed in mission critical applications.

To address the problem of technique integration for power-aware embedded systems, we propose a new design tool framework called IMPACCT and the associated design methodology. The system modeling methodology includes application model for capturing timing/power constraints and mode dependency at the system level. The tool performs power-aware scheduling and mode selection to ensure that all timing/power constraints are satisfied and that all overhead is taken into account. IMPACCT then synthesizes the implementation targeting a symmetric multiprocessor platform. Experimental results show that the increased dynamic range of power/performance settings enabled a Mars rover to achieve significant acceleration while using less energy. More importantly, our tool correctly combines the state-of-the-art techniques at the system level, thereby saving even experienced designers from many pitfalls of system-level power management.

**Keywords:** System-level power management, power-aware scheduling, power mode selection.

**Abbreviations:** DVS – dynamic voltage scaling; DPM – dynamic power management

## 1. Introduction

Recent years have seen the emergence of *power-aware* embedded systems. They are characterized by not only low power consumption, but more generally by their ability to support a wide range of power/performance trade-offs. These systems can be viewed as providing “knobs” that can be turned one direction to reduce power consumption or the other direction to increase performance. The ability to maximize the range of power/performance trade-offs is driven by new applications that demand very high performance while operating under stringent timing and power constraints. One such application can be found in the space domain in the form of a rover.



© 2002 Kluwer Academic Publishers. Printed in the Netherlands.

Let us consider the Mars Pathfinder rover from NASA/JPL [1]. It was designed to roam on Mars to take digital photographs and perform scientific experiments over several hundred days. Its energy sources consist of a battery pack and a solar panel, and future versions are expected to incorporate a nuclear generator or other energy scavenging devices. The initial version was designed to be low-power, and this was accomplished by serializing all tasks, including mechanical and heating as well as computation. However, low-power also means low performance in this case, as the rover could move at most 10cm per minute, and shoot and wirelessly transmit at most three high-resolution photos in a day. Even though during daytime the solar panel could output more power than could be consumed by the rover, the rover was unable to take advantage of this power; instead, the extra heat was redirected to heating the wheels.

This is an instance where a low-power design may be correct, but a power-aware version can do much better. We have proposed a power-aware version of the rover: by allowing power usage and performance to track power availability, the power-aware system with more system-level parallelism achieved 33% speedup while saving 33% battery energy [21].

Encouraged by the initial success, we explored additional power management opportunities at the system level. Since the goal is to increase the dynamic range of power/performance curves, we sought ways to increase performance in one direction and to reduce power in the other. To increase performance when more power (such as solar) is available, we attempted system-level task motion, a class of effective techniques that have been developed for many different domains ranging from VLIW instruction scheduling to hardware synthesis. To reduce energy consumption, we also attempted to incorporate other researchers' new power management techniques that are power-aware. These include a variety of dynamic voltage scaling (DVS) and scheduling algorithms for modern embedded processors, whose voltage and frequency can be controlled.

However, a somewhat surprising result was that many of these performance enhancement and power reduction techniques yield incorrect and rather counterintuitive results when applied together at the system level. Existing scheduling techniques that treat the power budget as a resource constraint (e.g., mapping power to the total register count) fail to correctly satisfy the power constraints. On the other hand, DVS techniques, which slow down processors in order to achieve quadratic energy savings, actually end up consuming more energy at the system level.

The main reason these techniques fail is that many important system-level dependencies are not properly modeled or considered. In a system, the components do not work independently; instead, they work very much together with each other, and power management decisions made on one component can have a chain of effects on the power usage of the other components. This

is further complicated by the fact that different components are built with different power management capabilities. In the Mars rover, not all components are power manageable. In fact, some components include motors for steering and driving the rover, heating elements for melting the frozen lubricants on the wheels, and the R/F module. Many of these components cannot scale their voltage or frequency the same way a processor can. Furthermore, mode changes are seldom instantaneous or free; instead, they incur nontrivial timing and power overhead that cannot always be amortized. As a result, the combined effect of these power management techniques can often contradict the designer's intuition and even cancel each other's effects.

It is clear that an integrated design tool is sorely needed to help designers manage such a multi-dimensional problem: functional correctness, timing constraints, and power awareness. To address these difficult problems, we develop a tool-based design methodology called IMPACCT, for Integrated Management of Power-Aware Computing and Communication Technologies. As with most system-level design tools, IMPACCT starts with high-level modeling of the application, separate from the target architecture. The designer then uses IMPACCT to transform and refine the high-level model towards implementation. IMPACCT also supports power-aware functional simulation to help with design validation.

This paper focuses on two of the core design tasks in IMPACCT: power-aware scheduling and mode selection. The objectives are to enhancing the power/performance trade-off range and to correctly compose different component level power management techniques at the system level. Power and timing constraints can be used as knobs to tune the system for performance or power, without hardwiring to either goal. To maximize performance and resolve power "hot spots," we exploit system-level task motion under pair-wise timing and total power as constraints. A distinguishing feature of our work is our ability to handle *co-activation*, an essential property for the correct operation of these embedded systems. Furthermore, we propose mode selection as a generalized way for fully exploiting novel power management features provided by an increasingly intelligent class of power-aware components. They are capable of managing power and provide many more *power modes*. However, today's power management techniques often cannot take full advantage of these rich features, but instead they use only two or three modes (e.g., on/off). Our mode selection methodology models the dependency and produces a mode schedule that considers restricted transitions and overhead amortization. Together these techniques not only form the foundation for integrating many power management techniques, but more importantly they help even experienced designers avoid many pitfalls with composing these components at the system-level.

In the next section, we review work related to power management and codesign and illustrate the pitfalls with applying today's techniques at the

system level. Section 3 provides an overview of IMPACCT including specification, architecture, and simulation. The sections that follow will describe scheduling and mode selection, and a summary of results for several real-life driving examples.

## 2. Related Work

To maximize the power/performance range in power-aware systems, we can draw from many techniques developed for low power and high performance. Low power can be achieved by shutting down idle components, changing mode, or scaling the voltage. In the other direction, high performance can be achieved by various pipelining techniques done for VLIW and high-level synthesis. This section surveys related works in these areas with a discussion on their integration at the system level.

### 2.1. LOW-POWER TECHNIQUES

Many low-power techniques have been developed at all levels, ranging from circuit and logic levels and micro/macro-architectural levels to operating system and application levels. For system-level designs, since the components are largely off-the-shelf or already designed, the applicable techniques include dynamic voltage scaling (DVS) and dynamic power management (DPM) with subsystem shutdown.

#### 2.1.1. *Dynamic Voltage Scaling (DVS)*

DVS techniques have been developed for variable-voltage processors. Introduced by [43], with follow-up by [26, 11, 27] and more, DVS can achieve significant energy saving while still enabling the processor to continue making progress. Lowering the voltage will also require reduction in frequency, which has the effect of reducing dynamic switching power. Although DVS means running slower, today's techniques typically slow down just enough without violating timing constraints, and many are based on real-time task scheduling cores [10, 34, 35, 32]. It has been shown that maximal energy saving is achieved by running the processor at the slowest possible constant speed, rather than running tasks at full processor speed and changing the processor to a lower power mode when idle [4]. Hong et al [10] proposed a heuristic for scheduling real-time tasks on a single variable voltage processor. Shin [34] exploited both execution time variation and idle time intervals for fix-priority tasks. Shin's algorithm in [35] determines the lowest maximum processor speed for each job to achieve power reduction. Quan and Hu [32] further greedily determine the lowest voltage for a set of tasks to achieve more energy savings.

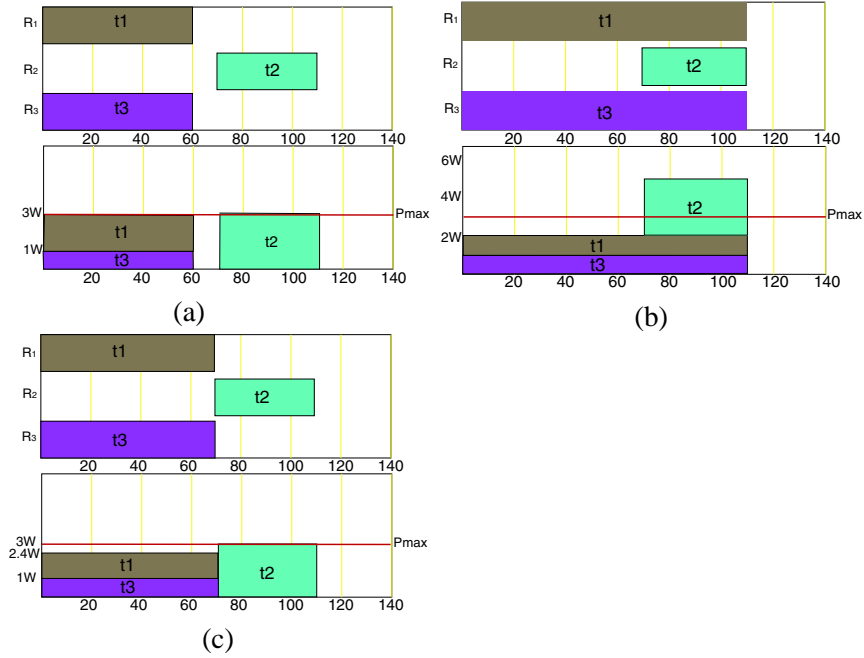


Figure 1. An application scenario that has resource dependency. (a) Initial schedule and power profile; (b) greedy voltage scaling results in a power spike that violate maximum power constraint; (c) a feasible solution that meets both power and timing constraints, and saves energy as well.

However, DVS’s notion of a system is currently limited to a single processor without considering peripheral devices, and the results are not generalizable to multiple processors. What these DVS techniques have in common is that they are greedy and assume a single processor. A power-aware embedded system, however, consists of multiple *resources*, which may be one or more processors and peripheral devices. Luo and Jha [22, 23] presented static scheduling for multiple processing elements (PEs) by reordering tasks and applying voltage scaling in this post-processing step to smooth the system-level power profile. Unfortunately, all of these greedy DVS techniques fail to generalize to multiple resources when there are co-activation dependencies and power constraints, as shown in the following example.

*Example: (DVS fails in multi-resource)*

Fig. 1(a) shows a Gantt chart (on top) and the corresponding power profile (bottom) for a system with three resources:  $R_1$  is capable of voltage scaling, while  $R_2$  and  $R_3$  are not voltage scalable. The task  $t_1$  on  $R_1$  has a deadline at time 110. The system also has a maximum power constraint of 3W. Furthermore, the behavior of the application dictates that  $R_1$  and  $R_3$  be *co-active*. Co-activation means the operation of one resource requires the power

| Schedule           | Timing violation | Power violation | Energy cost |
|--------------------|------------------|-----------------|-------------|
| Initial, Fig. 1(a) | No               | No              | 300         |
| Greedy, Fig. 1(b)  | No               | Yes             | 320         |
| Optimal, Fig. 1(c) | No               | No              | 288         |

Figure 2. Comparison of three schedules.

consumption of other dependent resources. A simple example is that when the CPU is running, it imposes a co-activation dependency on the memory, but co-activation can be much more general as imposed by tasks. We assume the starting time of each task is fixed in this example.

Fig. 1(b) shows the schedule (top) and the power profile (bottom) obtained by greedily slowing down  $R_1$  to achieve energy saving. Even though all timing constraints are satisfied, it violates power constraints and its energy consumption is not minimum. The max power constraint is violated because when task  $t_1$  is stretched out, and when it overlaps task  $t_2$  during the time interval from 70 and 110, their total power exceeds the max power constraint. Due to the co-activation dependency between  $R_1$  and  $R_3$ , the energy saving by  $R_1$  due to voltage scaling is more than offset by  $R_3$  whose voltage is not scalable (as given), but its execution is prolonged by  $R_3$ .

The optimal schedule and power profile are shown in Fig. 1(c). Resource  $R_1$  is slowed down without overlapping task  $t_2$  on Resource  $R_2$ . This way, max power is not violated. Although task  $t_3$  on resource  $R_3$  is stretched with task  $t_1$  and therefore consumes more energy than it does in Fig. 1(a),  $t_1$  saves even more energy due to voltage scaling of resource  $R_1$ . As a result, the system achieves minimal energy while satisfying all constraints. Fig. 2 summarizes the energy costs.

Another problem not highlighted with this example is that mode changes may incur nontrivial power or timing overhead. If so, the overhead must be considered in determining the feasibility of the revised schedule.

### 2.1.2. Dynamic power management (DPM)

DPM techniques, which are based on timeout or event prediction, assume multiple components with multiple modes. Subsystem shutdown decision can be based on fixed idle time, adaptive timeout, or profile and runtime history [39, 37, 6, 2]. The simplest power management policy is time-out with a fixed or predicted amount of time before the system's shutdown or power-up [40, 12]. Stochastic models [3, 30] are used to address the uncertainty in system behaviors. Simunic et al [36] combines stochastic-modeled power man-

|                      | DPM      |         | DVS              |          | MS |
|----------------------|----------|---------|------------------|----------|----|
|                      | [40, 12] | [3, 30] | [10, 34, 35, 32] | [22, 23] |    |
| Timing as constraint | N        | N       | Y                | Y        | Y  |
| Power as constraint  | N        | N       | N                | N        | Y  |
| Timing overhead      | Y        | Y       | N                | N        | Y  |
| Power overhead       | Y        | Y       | N                | N        | Y  |
| Multiple resources   | N        | Y       | N                | Y        | Y  |

Figure 3. Comparison of dynamic power management (DPM), dynamic voltage scaling (DVS), and mode selection (MS).

agement with dynamic voltage scaling to achieve significant power reduction in portable systems.

DPM techniques can be effective for minimizing energy and time penalties on average, but they have several limitations. First, most treat either power or timing as an *objective* or penalty, rather than a *constraint*. In real systems, the max power is a real, hard constraint, whose violation can lead to malfunction. Max power was not of central concern previously, but as we consider additional power sources such as solar whose maximum output can vary, they must be strictly satisfied. This becomes especially important as we increase the dynamic range of power by increasing parallelism. Second, they have not considered inter-component dependency in a system, with the exception of Qiu, Qu and Pedram in [31]. Their Generalized Stochastic Petri Net (GSPN) can capture some dependencies in client-server applications. However, only one server is modeled to process an incoming request.

Fig. 3 summarizes the features of the techniques surveyed here. The last column shows our new approach, mode selection, which combines the advantages of existing approaches. It is constraint-driven, enabling us to make power/performance trade-offs without hardwiring specific goal or policy in the design.

## 2.2. HIGH PERFORMANCE THROUGH PIPELINE TRANSFORMATION

In addition to low power, dynamic range can also be increased towards high performance by drawing from works on retiming and rotation and applying them to the system level. Leiserson et al [16] first established the theoretical foundation for retiming synchronous circuits, and this has been extended to loop pipelining and scheduling for VLIW processors [33, 5, 13]. Shifting or

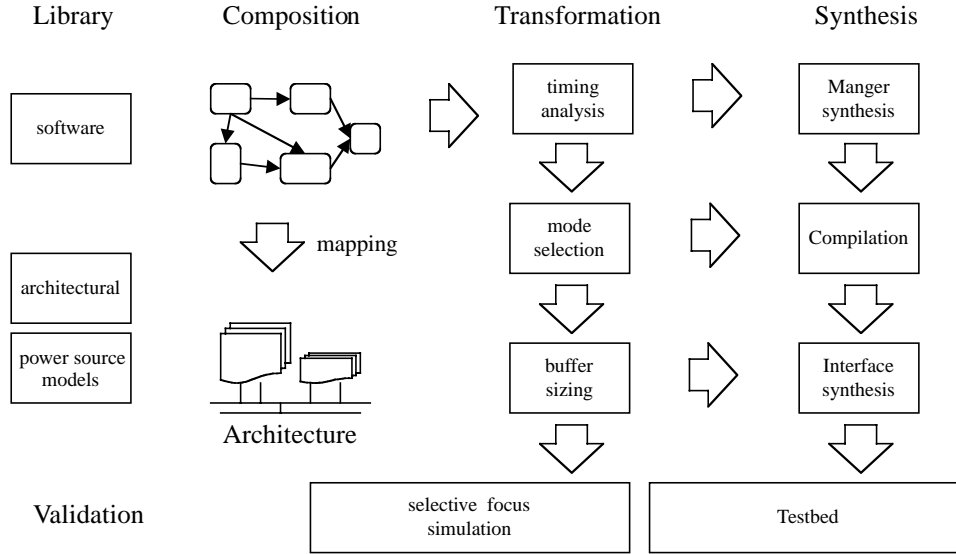


Figure 4. The IMPACCT system-level design tool for power-aware embedded systems.

“rotating” tasks in a data flow graph (DFG) across the iteration boundary can result in a shorter execution time or alleviate the resource pressure (e.g. number of registers and functional units). Such techniques are also used in power minimization by reducing switching activities [15, 44].

Existing techniques need significant enhancements before they can be correctly applied to our system-level power management problem. The main reason is that new types of dependencies must be modeled at the system level, and they cannot be readily handled by existing techniques by preprocessing. The tasks to be scheduled are related to each other not only by precedence or data dependency and timing, but also co-activation dependency as mentioned earlier. Preprocessing by grouping co-activated tasks will not yield correct results due to the presence of timing constraints and lack of interchangeability in resources.

### 3. Overview of IMPACCT

IMPACCT is a system-level design tool for exploring power/performance trade-offs in hard real-time systems by means of power-aware scheduling and architectural configuration. The current implementation includes an interactive graphical tool for scheduling, mode selection, and an interface to a simulation back-end for integrated evaluation of the system under design. 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

an entire system. Furthermore, a system in the broad sense includes not only computational components but also those in the non-computational domains (e.g., mechanical and thermal subsystems), which are equally important in many mission-critical applications. IMPACCT is the first tool to correctly address all of these system-level power management issues. Fig. 4 shows the main components of the IMPACCT framework, and this section will highlight each box in order. The combination of these features in IMPACCT presents a compelling design-time tool for engineers to explore a wide range of system-level power/performance trade-offs with confidence.

### 3.1. INPUT: APPLICATION MODEL AND CONSTRAINTS

To use IMPACCT, the designer must construct a model for the application and constraints. Although the detailed application behavior is ultimately written in one of the system programming languages (such as C, C++, Ada, Java, etc), IMPACCT does not process these files directly; instead, they are passed to power/timing analysis or simulation tools for estimation or validation. IMPACCT expects the designer to construct a higher-level model for the application in our custom language. This description includes ports and channels for expressing data dependencies, and it supports timing and power constraints. Note that timing and power are not necessarily intrinsic to the application problem itself, but they should really be viewed as “budgets” whose values are selected based on engineering decisions. One main purpose of the tool is to help designers with constraint refinement or adjustment (re-budgeting) by giving them a quick estimate. This approach allows the designer to start working with the power/timing budget for various tasks to be performed long before the program or component is designed. As these pieces become available, they will then be used to refine these budgets with more accurate estimation.

We currently support power constraints and timing constraints. *Power constraints* are the min/max bounds on the power dimension of the power profile. The *max-power* constraint requires that the system never draw more than the specified amount of power at any given moment. It may be derived from the maximum current rating of the power supply and can be a hard constraint. Even though most systems to date could assume sufficient power by design, the next generation power-aware embedded systems will need to work with a much more diverse set of power sources with much lower power budgets and reduced availability. This will make max-power a hard constraint. On the other hand, we believe *min-power* will be an important constraint, especially for systems with renewable energy sources. One reason is that as heat becomes an important issue in embedded systems, unused power must be carefully dissipated or else the system risks overheating. Rechargeable batteries have finite capacities and will contribute to the heat when overcharged.

In the case of the rover, it would require extra heat dissipation hardware to handle unused min-power. This would add extra weight and cost to the rover. Another reason is that the min and max constraints together will be a way to explore power/performance trade-offs without being hardwired to the low-power goal. Both min and max power constraints may be functions over time.

Timing constraints are in the form of *min/max timing separation* between pairs of events, where an event can be the start or end of a task. This is a general way for expressing precedence, absolute and relative deadlines, and also co-activation. Tasks assigned to different resources may run in parallel. We currently use a simple custom language to capture these timing constraints. The syntax of this high-level file is not important; it just has to be expressive enough to construct a graph description of the pair-wise timing constraints.

### 3.2. TARGET ARCHITECTURE AND MAPPING

The input to IMPACCT consists of a model for the target architecture and application-to-architecture mapping. The target system architecture provides the primitives for power management as well as the power/timing attributes needed for scheduling and mode selection. The elements of the application model are mapped to those of the target architecture: that is, the tasks are mapped to the processors, and channels mapped to the busses. IMPACCT provides a *component library* and a *system architecture template* to aid the description of the target architecture.

The component library consists of models for components and busses in the target architecture. They include processors, memory modules, bus controllers, communication modules, sensors and actuators, digital cameras, and various peripheral devices. The designer instantiates and configures these components from the library. The component models will provide an interface for the rest of the design tool to ask questions about the power/timing attributes needed to synthesize or for simulation. Some of these attributes such as modes, clock rates, or voltage may be stored as fixed values, but others such as the execution delay or the power consumption may need to be derived by either evaluating a formula or by simulation. Each component model may encapsulate any number of detailed models (RTL, SPICE, power-macromodel), but they are abstracted from the designer. IMPACCT augments these low-level models with higher-level models for supporting system-level power management. These features include the power modes, the allowed transitions between modes, the power/timing coefficients associated with each mode and the transitions, and the interface description for controlling these power management features. This mode model will be described in more detail in the Mode Selection section.

Unlike traditional hardware/software co-design that is more about free-form exploration of an optimal architecture, we take a platform-based approach for practical reasons. IMPACCT provides architectural templates for configurable platforms, and currently supported is a symmetric multiprocessor architecture interconnected with a two-tier bus. It can be configured for different numbers of processors and components from the library. The two-tier bus includes the IEEE 1394 (“FireWire”) for high-speed, real-time data and the I<sup>2</sup>C for low-speed control. Both are power efficient and support dynamically adding/removing or powering up/down individual nodes for the purpose of power management. The software runs on WindRiver vxWorks, a commercial real-time OS for embedded systems. Scheduling and mode selection are performed statically but the run-time system can switch between different schedules.

### 3.3. POWER-AWARE SCHEDULING

Our scheduler enhances the dynamic range of power/performance trade-offs. The core scheduler handles both timing and power as constraints, not just goals. Power and timing are both treated as min/max constraints. The advantage is that these constraints become the knobs for tuning the system’s power/performance trade-offs. By making the constraints track the available solar power, the IMPACCT scheduler has been shown to accelerate the system while saving energy at the same time for a Mars rover. This feature will be critical to also systems that use alternative energy sources such as thermal batteries as well as those with thermal management concerns. In addition, we explore system-level task motion as a way to vary the level of parallelism to further increase the dynamic range of these systems. Scheduling will be discussed in Section 4.

### 3.4. MODE SELECTION

Another complementary feature in IMPACCT is mode selection. It is the task of deriving a schedule for mode changes in the components of the system, such that all architectural effects are properly considered. It takes as input a schedule from the previous step, and it decides what power modes in which each component should operate over time. Mode selection addresses issues that fail to be handled by today’s greedy dynamic voltage schedulers by considering the transition overhead and dependencies. It will not change mode if the time/power overhead involved cannot be amortized over the tasks to be performed, and it also prevents system-level power spikes due to greedy, isolated voltage scaling. More importantly, IMPACCT’s mode selection properly models and handles co-activation dependencies. For example, when the processor is on, the memory must be on, too. By modeling these dependencies in

the mode selection step, IMPACCT will ensure that the resulting power management policy considers all features critical to the correct operation of the entire system. Mode Selection will be described in more detail in Section 5.

### 3.5. SIMULATION SUPPORT

IMPACCT supports simulation at various stages of the design flow. The high-level application description can be simulated functionally without mapping to an architecture. The IMPACCT high-level simulator has been integrated into the scheduler. It not only computes the ordering of the tasks to run on generic resources, but also invokes the compiled application files via native calls to simulate their functionality. The high-level simulator is also responsible for implementing the inter-process communication mechanisms using buffer management.

This setup also enables the integration of heterogeneous simulation and emulation models with a uniform user interface. Because the simulation models are externalized, the IMPACCT simulation coordinator can replace the external, native calls with any other calls, as long as they conform to a compatible application programming interface. For example, hardware-in-the-loop simulation can be accomplished by replacing these external calls with calls to device drivers that control emulation hardware. Similarly, these calls can also be made to detailed simulation models when accuracy or controllability is required. The back-end is completely decoupled from the front-end, which provides a uniform user interface including visualization support.

## 4. Power-Aware Scheduling

IMPACCT provides comprehensive scheduling support for maximal trade-offs between power and performance. We have developed static scheduling algorithms for satisfying timing and power constraints [20, 21], which varies the amount of parallelism at the system level according to the available power (e.g., from the solar panel). In addition, several extensions to the core scheduler have been implemented, and they further widen the trade space by means of system-level task motion [19] and aggressive mode selection. A number of task motions are not only effective for smoothing out the workload by borrowing time and power across iterations, but also enable additional mode change involving additional processors. This section summarizes the current state of the scheduler.

### 4.1. SCHEDULING UNDER POWER AND TIMING CONSTRAINTS

IMPACCT's core scheduler solves the power/timing-constrained problem as an instance of a graph problem. The timing constraints are modeled with

a **constraint graph**  $G(V, E)$ , where the vertices  $V$  represent tasks, and the edges  $E \subseteq V \times V$  represent timing constraints between tasks. Each vertex  $v \in V$  has three attributes,  $d(v)$ ,  $p(v)$  and  $r(v)$ , representing task  $v$ 's *execution delay*, *power consumption*, and *resource mapping*, respectively. Each edge  $(u, v) \in E$  has two attributes,  $\delta(u, v)$  and  $\lambda(u, v)$ .  $\delta(u, v)$  specifies the *min/max timing constraints* on the start times assigned by the function  $\sigma$  to tasks  $u$  and  $v$ , such that  $\sigma(v) - \sigma(u) \geq \delta(u, v)$ . If  $\delta(u, v) \geq 0$ , edge  $(u, v)$  is called a *forward edge* that specifies a *min timing constraint*. If  $\delta(u, v) < 0$ , it is a *backward edge* indicating a *max timing constraint*.  $\lambda(u, v)$  is called the *dependency depth*, which specifies constraints across iterations. An *iteration* is a full pass of executing of each of the tasks once in a valid order.  $\delta(u, v)$  and  $\lambda(u, v)$  indicate that the execution of task  $u$  in iteration  $i$  must precede task  $v$  in iteration  $i + \lambda(u, v)$  by  $\delta(u, v)$  time units. If  $\lambda(u, v) = 0$ , edge  $(u, v)$  specifies an *intra-iteration constraint*. Otherwise, it is an *inter-iteration constraint*.

A **schedule**  $\sigma$  assigns a start time  $\sigma(v)$  to each task  $v \in V$ . It has a *finish time*  $\tau_\sigma$  when all tasks complete their execution. Schedule  $\sigma$  is called *time-valid* if all the start time assignments do not violate any timing constraints, and tasks that share the same resource are serialized. If  $G$  represents an iteration of a loop,  $\sigma$  must also satisfy inter-iteration constraints such that they must hold across iterations when multiple copies of  $\sigma$  are concatenated.

A schedule  $\sigma$  has a **power profile** function  $P_\sigma(t)$ ,  $0 \leq t \leq \tau_\sigma$ , representing the instantaneous power consumption of all tasks during the execution of  $\sigma$ . The power profile is constrained by two parameters:  $P_{max}, P_{min}$ , such that  $P_{max} \geq P_\sigma(t) \geq P_{min} \geq 0$ . The **max power** constraint  $P_{max}$  specifies the maximum budget of supply power that can be provided by the power sources. The **min power** constraint  $P_{min}$  specifies the level of power consumption to maintain a preferred level of activity.

The max power constraint is a hard constraint. At any given time  $t$ , the value of the power profile function  $P_\sigma(t)$  must not exceed  $P_{max}$ . Schedule  $\sigma$  is called *power-valid* (or simply, *valid*) if it is time-valid and its power profile does not exceed the max power constraint. However, we treat the min power constraint as a soft constraint that could be violated occasionally in a valid schedule.

In cases where the min power constraint  $P_{min}$  represents the free power level, the energy drawn from the non-renewable energy sources is defined as the *energy cost*  $Ec_\sigma(P_{min})$  of a schedule  $\sigma$ . It distinguishes between costly power and free power in such a way that any power consumption below the free power level does not contribute to the energy cost on non-renewable energy sources, and therefore should be utilized maximally.

Power-aware scheduling algorithms are presented in Fig. 5, 6, 7. The goal is to find a schedule  $\sigma$  that is both time-valid and power-valid. We propose an incremental approach by solving one type of constraints at a time in three steps. Since this is an NP-hard problem, we also have developed slack-based

```

TimingScheduler(Graph  $G$ , vertex  $anchor$ , vertex  $c$ )
   $L(c) := \text{SINGLE SOURCE LONGEST PATH}(G, anchor)$ 
  if (positive cycle found) then return FAIL
  if ( $Succ[c] = \emptyset$ ) then return  $\sigma$  with  $\sigma(c) := L(c)$ 
  while ( $Succ[c] \neq \emptyset$ ) do
     $v := \text{extract one vertex from } Succ[c]$ 
  B:    $Succ[v] := Succ[v] \cup Succ[c]$ 
       foreach (vertex  $u$  that is not traversed) do
         if ( $r(u) = r(c)$ ) then serialize  $u$  after  $c$ 
        $\sigma = \text{TimingScheduler}(G, anchor, v)$ 
       if ( $\sigma \neq \text{FAIL}$ ) then return  $\sigma$  with  $\sigma(c) := L(c)$ 
       undo changes to  $G$  since step B
  return FAIL

```

Figure 5. Algorithm for timing scheduling

heuristics that steer the scheduler towards quickly finding feasible solutions by localized task movements instead of expensive backtracking. First, based on the constraint graph of the problem, we apply the timing scheduler (Fig. 5) to find a schedule that is time-valid. Second, the max power scheduler (Fig. 6) tries to satisfy the max power constraint using slack-based heuristics. Finally, given a valid schedule provided by the previous step, the min power scheduler (Fig. 7) refines the schedule to improve the utilization of the min power level. Details of the scheduling algorithms and heuristics can be found in [21].

#### 4.2. DYNAMIC RANGE ENHANCEMENT THROUGH TASK MOTION

To extend our power-aware scheduling to iterative tasks, we present power-aware task motion as an effective way to enhance the dynamic range of power/performance trade-offs. The distinguishing feature of our method from previous scheduling techniques is that existing techniques either do not have timing constraints  $\delta(u, v)$  in their dataflow graphs (DFG), or the value of  $\delta(u, v)$  is always 0 or 1 that indicates only precedence (data dependency). In addition, we propose a new class of timing constraints called *utilization-based* constraints to enable more aggressive, domain-specific transformation for preemptible background tasks or non-computational tasks.

Power-aware task motion is performed in two steps: (a) transforming the problem into its variations by task motion, and (b) power-aware scheduling for each variation. We first construct a timing constraint graph  $G(V, E)$  that expresses the tasks in an embedded system with pair-wise constraints. By applying task motion, tasks are moved across iteration boundaries such that intra-iteration and inter-iteration constraints are converted to each other.

Without loss of generality, we focus our discussion on task *promotion* by which the execution of a task is shifted to the previous iteration of the loop,

```

MaxPowerScheduler(Graph  $G$ , vertex  $anchor$ ,  $P_{max}$ )
 $\sigma :=$  TimingScheduler( $G$ ,  $anchor$ ,  $anchor$ )
if ( $\sigma = \text{FAIL}$ ) then return FAIL
for ( $t := 0$ ;  $t \leq \tau_\sigma$ ;  $t := t + 1$ ) do
   $S :=$  set of all active tasks at  $t$ , ordered by  $slack_\sigma$ 
   $reschedule := \text{FALSE}$ 
  while ( $P_\sigma(t) > P_{max}$  or  $reschedule = \text{TRUE}$ ) do
B:   repeat
      $v := \text{EXTRACT MAX}(S)$ 
     if ( $slack_\sigma(v) = 0$ ) then  $reschedule := \text{TRUE}$ 
     delay  $v$  by some time units (heuristically determined)
     until ( $P_\sigma(t) \leq P_{max}$ )
     if ( $reschedule = \text{TRUE}$ ) then
       lock start times of remaining tasks in  $S$ 
        $\sigma := \text{MaxPowerScheduler}(G, anchor, P_{max})$ 
       if ( $\sigma \neq \text{FAIL}$ ) then return  $\sigma$ 
       undo added edges to  $G$  since step B
  return  $\sigma$ 

```

Figure 6. Algorithm for max power scheduling

```

MinPowerScheduler(Graph  $G$ , vertex  $anchor$ ,  $P_{max}, P_{min}$ )
 $\sigma := \text{MaxPowerScheduler}(G, anchor, P_{max})$ 
if ( $\sigma = \text{FAIL}$ ) then return FAIL
if ( $\rho_\sigma(P_{min}) = 1$ ) then return  $\sigma$ 
 $\sigma_{old} := \sigma$ 
while ( $\text{TRUE}$ ) do
  for ( $t$  in a heuristic order of range  $[0, \tau_\sigma]$ ) do
    if ( $P_\sigma(t) < P_{min}$ ) then
       $S :=$  set of tasks that start before  $t$ 
      foreach ( $v \in S$  such that  $slack_\sigma(v) \geq t - \sigma(v) - d(v)$ ) do
B:    $\sigma_{new} :=$  delay  $v$  some time units such that  $v$  is active at  $t$ 
      if ( $\sigma_{new}$  is valid and  $\rho_{\sigma_{new}}(P_{min}) > \rho_\sigma(P_{min})$ ) then
         $\sigma = \sigma_{new}$ 
        if ( $\rho_\sigma(P_{min}) = 1$ ) then return  $\sigma$ 
      else
        undo added edges to  $G$  in step B
  if ( $\sigma = \sigma_{old}$ ) then return  $\sigma$ 
return  $\sigma$ 

```

Figure 7. Algorithm for min power scheduling

and the instance of the same task in the next iteration is included into the new loop body. The procedure for *demotion* can be similarly defined.

When a task  $v$  is being promoted, its corresponding min timing constraints (zero or positive values) will become max timing constraints (negative values); and vice versa, its corresponding max timing constraints will transform into new min timing constraints. Promotion effectively reduces the values of min constraints and makes the problem easier to solve by exposing more scheduling opportunities. We say that the constraint is *relaxed*, and this is a key technique for increasing the system's dynamic range.

Task motion is based on the classification of intra-iteration and inter-iteration timing constraints. However, in some cases, it is difficult or unnecessary to decide whether a timing constraint should be intra-iteration or inter-iteration. Such cases are present in the Mars rover. For example, the timing constraints between a heater and a motor may be modeled as either intra-iteration or inter-iteration, since the heaters and the motors need not stay in the same iteration. In the computation domain, these correspond to background, preemptible tasks that need not synchronize with the main control loop but must be given a share of the CPU time to avoid starvation. In real-time systems literature, this is called share-driven scheduling.

We call such constraints *utilization-based* timing constraints. They can be expressed as either intra-iteration or inter-iteration ones. A utilization constraint between two tasks  $u$  and  $v$  is also represented as an edge  $(u, v) \in E$  in constraint graph  $G$  with its dependency depth denoted as  $\lambda(u, v) = *$ , indicating that it can be either zero or non-zero; and it will be included in iteration graph  $G'$  initially. Utilization-based constraints will always be relaxed to increase scheduling opportunities. Detailed discussion of the power-aware task motion technique can be found in [19]. The increased schedulability may come at the price of increased jitter, and this may not be acceptable to all applications. In such cases, the designer would not use utilization constraints but would use inter-iteration min-separation constraints instead to control jitter. Alternatively, buffering can also smooth out jitter if the latency constraint allows so.

## 5. Mode Selection

As new components are being built with more intelligent power management features, it will be important to take full advantage of these features. Unlike yesterday's components that offer only very simple modes such as on, off, and sleep, new components can actually be an entire system on a chip and can have more than a dozen modes. For example, today's typical hard disk offers over fifteen power modes. Unfortunately, most of today's power management techniques are able to handle only two or three modes. We propose mode

selection as a systematic way for optimizing the power/performance settings at the component level to best match the execution context of the system. It is general because it subsumes voltage scaling and it can also encompass techniques at other levels of abstraction, including selection of alternative algorithms and data structures as new ways of managing power. The key problems in mode selection include (1) modeling the attributes associated with the modes and transitions between pairs of modes; (2) capturing the interdependency between modes of different components in a system; (3) defining the cost function based on the power-delay product; and (4) efficiently generating a progression of these mode settings while satisfying all timing and power constraints.

### 5.1. POWER MODES

A component is a container of resources, and each resource may be operable in a set of power modes. We will use the terms components and resources interchangeably without ambiguity when convenient. A resource  $\gamma$  is defined as a graph  $R_\gamma(M_\gamma, H_\gamma)$ , where  $M_\gamma$  is a set of vertices, and  $H_\gamma \subseteq M_\gamma \times M_\gamma$  is a set of edges. A vertex  $m \in M_\gamma$  represents a power mode of resource  $\gamma$ . An edge  $(m, n) \in H_\gamma$  represents a mode change from mode  $m$  to mode  $n$ . Power consumption is characterized by a function,  $P$ , mapping from a power mode to a power number. Formally,  $P : M_\gamma \rightarrow \mathbb{R}^+$ . We also define a matrix for the timing and energy overhead on mode changes. If a particular mode change is not allowed, its entry in the overhead matrix would be  $\infty$ .

For example, a processor may have active, idle, and sleep modes. Changing from any mode to other modes incurs time and energy overhead. Waking up from sleep to active mode needs more time and energy than going from idle to active mode. At the architectural level, mode change overhead may be time and energy of changing hardware configurations only. At the application level, it may include overhead to restore the context, reinitialize the OS or load a program. The modes are not limited to simple power modes such as on, off, doze, nap, or sleep, but also encompass cache configurations for processors, different encoding/decoding techniques for a radio, variations of compiling techniques for software components, and different transmission protocols for bus drivers.

Given  $N$  resources  $(\gamma_1, \gamma_2, \dots, \gamma_N)$ , a configuration is a combination of  $N$  modes, in the form of  $(m_1, m_2, \dots, m_N)$ . The choice of modes at any moment is driven by the tasks that are running on the resources at the time, subject to the power and timing constraints. The output of mode selection is a sequence of configurations whose total energy is minimized.

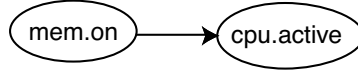


Figure 8. Mode dependency example: the memory is on only if the CPU is in active mode.

## 5.2. MODE DEPENDENCY GRAPH

Selecting (or not selecting) a mode of a resource may impact the modes that other resources are allowed to select. The impact may be imposed by co-activation, which forces another resource to select a corresponding mode; it may also be exclusion, enabling, and many other possible types of dependencies. These dependencies may be extracted from application level specifications or policies for safety, security, fault-tolerant, power reduction, or may be explicitly specified as mode correlation [45]. In any case, a *legal* configuration is a mode combination for the resources such that all mode dependencies are met when a given set of tasks is running; a *feasible* configuration is one that is legal and satisfies all the constraints (namely timing and power). We use a data structure called the mode dependency graph (MDG) that enables efficient generation of legal configurations. This is done in a heuristic order that facilitates the search for feasible configurations.

An MDG  $G_m(V, E)$  represents the inter-resource dependencies, where a vertex  $v \in V$  is a resource mode, and a directed edge  $e \in E$  connects two vertices. Each vertex is represented by a circle with a label in the format of “*res.mod*,” where *res* is the resource and *mod* is the mode of the resource. If two vertices have the same labels, we considered them identical.

The *value* of a vertex  $|v|$  is defined as:

$$|v| = \begin{cases} True & \text{if } res \text{ is in } mod \text{ mode,} \\ False & \text{if } res \text{ is in other mode,} \\ Undetermined & \text{if } res \text{ has not been selected a mode.} \end{cases} \quad (1)$$

An edge in the MDG represents dependency between two modes. Suppose an edge  $(u, v) \in E$ ,  $u = res1.mod1$ ,  $v = res2.mod2$ . The two modes *mod1* and *mod2* satisfy the mode dependency graph if  $|u|$  is *True* **only if**  $|v|$  is *True*. For example, we can represent the dependency between a CPU and a memory chip such that the memory is on only if the CPU is in active mode (see Fig. 8). If the CPU is in sleep mode and the memory is on, then it violates the mode dependency. In addition, we also support logical operators as another kind of vertices. An operator vertex is represented by a square with an operator label in it. An operator vertex  $v_{op}$  has at least two fan-in, and at least one fan-out, meaning that  $v_{op}$  has at least two vertices pointing to it and it points to at least one vertex.

| AND |   |   | OR |   |   | XOR |   |   |
|-----|---|---|----|---|---|-----|---|---|
| A   | B | C | A  | B | C | A   | B | C |
| T   | T | T | T  | T | T | T   | T | F |
| T   | F | F | T  | F | T | T   | F | T |
| F   | T | F | F  | T | T | F   | T | T |
| F   | F | F | F  | F | F | F   | F | F |
| X   | U | U | X  | U | U | X   | U | U |
| U   | X | U | U  | X | U | U   | X | U |

Figure 9. Truth table for *AND*, *OR*, and *XOR* operators. C is the output of  $A \text{ op } B$ . T: True; F: False; X: don't care; U: undetermined.

The value of an operator vertex can be obtained by evaluating the logic functions that the graph represents. We define the operators *AND*, *OR*, *XOR*, and *MUTEX*. The truth tables of operator *AND*, *OR*, and *XOR* are listed in Fig. 9. The meaning of *AND*, *OR* and *XOR* follows the normal boolean functions in the same names except when any input is “undetermined,” the output is “undetermined.” A binary *MUTEX* (not listed in Fig. 9) is the same as an *XOR*; when *MUTEX* has multiple inputs (more than two), the output is *True* if and only if one input is *True* and the rest of input are *False*.

The extended mode dependency graph also follows “only-if” interpretation. For example, the top left item in Fig. 10(a) shows the dependency among a sensor (S), a radio device (R) and a processor (A). The semantics expressed here is that S and R are both off only if A is in sleep mode. If, for example, S is off, R is off, and A is in active mode, then it violates the dependency. If S is on, R is off, and A is in sleep mode, it does not violate the dependency.

Fig. 10(a) shows the mode dependency graph of the microsensor example [38]. The microsensor system consists of a sensor to sense the environment, a processor, a memory module, and a radio modem. In this system, the behaviors and dependencies of the devices can be derived from high-level power management policies: the sensor and the radio are both off only if the processor is in sleep mode; either of them is on only if the processor is in sleep mode or idle mode; both the sensor and the radio are on only if the processor is in active mode; the memory is on if and only if the processor is active.

### 5.3. COST FUNCTION

We define the cost function in two main parts: operation energy  $E_{op}$  and transition energy  $E_{tr}$ . We have  $E_c = E_{op} + E_{tr} = P_{op}T_{op} + P_{tr}T_{tr}$ . Operation energy

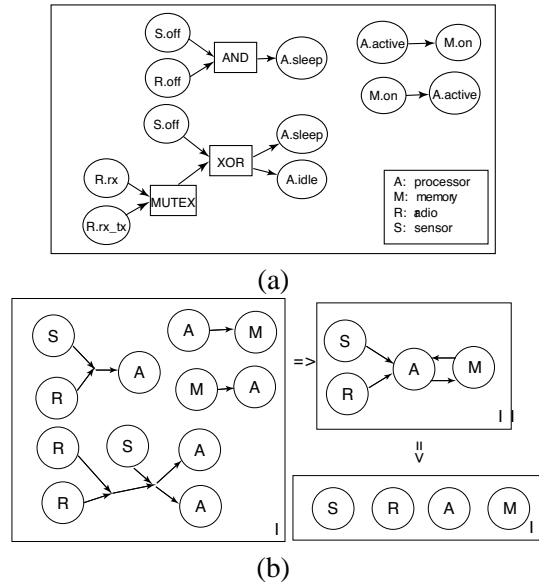


Figure 10. (a) The mode dependency graph for a micro sensor. (b) Reduce an MDG to a resource list for generating legal configurations. Box I: from the MDG, shrink each operator vertex to a point, and remove mode name in each mode vertex. Box II: remove the redundant vertices and edges. Box III: break the cycle by removing one edge in the cycle, and apply topological sort to obtain a resource list.

is the energy consumed when the component stays in a mode, and transition energy is the energy consumed when the component changes from one mode to another. Transition time is the delay on mode change. Sometimes either the transition power or transition time is large, which makes the transition cost high. For the purpose of computing cost, idle tasks are introduced later and included in the total operation energy. Our goal is to find a sequence of configurations that minimize the cost function.

#### 5.4. CONFIGURATION ENUMERATION

The mode dependency graph (MDG) captures enough information to enable the efficient generation of *feasible* configurations over time. The details of the configuration enumeration algorithm are in [17]. It is a special version of topological traversal on the MDG with backtracking. In practice, however, backtracking is not necessary because all of the infeasible configurations will be pruned out using the MDG. The search time is much closer to linear than exponential. This section illustrates configuration enumeration for a microsensor node in a distributed sensor network. It consists of a sensor, a processor, memory chips, radio frequency module and other auxiliary components. When the sensor obtains information from environment, it sends data

| component     | mode                |
|---------------|---------------------|
| sensor (S)    | on, off             |
| processor (A) | active, idle, sleep |
| memory (M)    | on, off             |
| Radio (R)     | rx, tx_rx, off      |

| mode | S   | R     | A      | M   |
|------|-----|-------|--------|-----|
| M1   | on  | tx_rx | active | on  |
| M2   | on  | rx    | idle   | off |
| M3   | on  | rx    | sleep  | off |
| M4   | on  | off   | sleep  | off |
| *M5  | off | tx_tx | active | on  |
| *M6  | off | rx    | idle   | off |
| *M7  | off | rx    | sleep  | off |
| M8   | off | off   | sleep  | off |

Figure 12. Legal configurations generated from MDG.

to the processor, the data is processed and sent to a base station or another network node via the RF module.

Fig. 11 summarizes the modes of the components. The sensor and the memory each has two modes, on and off. The processor has three modes, active, idle and sleep. The radio has three modes, receive-only (rx), transmit-and-receive (tx\_rx), and off. There are a total of 36 possible configurations for these components.

The inter-component relationships are specified in an MDG (more precisely, a forest) as shown in Fig. 10(a). The graph follows the “only-if” interpretation, and the graph semantics comes from system-level power saving policies. For example, the sensor and radio are both off only if the processor is in sleep mode. Either of them (not both) is on only if the processor is in sleep or idle mode. The processor is active only if the memory is on, and vice versa.

Using the MDG, our algorithm automatically generates eight legal configurations (see Fig. 12). These configurations can be considered by the system-level power manager for further pruning for feasibility, and energy optimization. This simple example demonstrates our algorithm’s ability to systematically generate legal configurations that satisfy inter-component dependencies. By editing the mode dependency graph based on the architectural features or application requirements, we can systematically obtain the new legal configurations. This pruning makes mode selection practical.

## 5.5. MODE SELECTION

Mode selection works as a post-processing stage after scheduling. It validates and improves the schedule with more architectural knowledge than the scheduler. Our approach considers resource/task dependencies and mode change

overhead, and tries to find a *mode schedule*, namely a sequence of feasible configurations, that not only satisfies system timing and power constraints but also minimizes energy.

Our search algorithm contains a loop with two steps. First we find modes for tasks that satisfy task dependencies and timing constraints. Second we determine modes for the idle intervals on each resource. Note that after the first step, the operation delay for certain tasks may be changed due to mode change (e.g., modes of different clock rate due to voltage scaling). We select modes for resources during task execution separately from idle intervals.

#### *Selecting modes for tasks*

We select modes for tasks by generating legal configurations and compute the timing and power for the tasks. Given a legal configuration, we can update the schedule since the operation delays of tasks become known based on their selected modes under co-activation dependencies. We check timing and power constraints for the new schedule. If it fails, we generate another legal configuration and check again; if successful, we use that configuration for mode selection of idle intervals.

#### *Selecting modes for idle intervals*

On each resource, overhead may be incurred on mode changes. We find a set of modes for each idle interval such that the time overhead of the mode change is less than the length of the idle interval. We decompose the revised schedule into *time intervals* within which no task is started or ended. We check system power constraints in each time interval. If the schedule fails to satisfy the power constraints, then we attempt a mode change on one or more resources that are currently in an idle interval, and we check power constraints again. If all the modes fail the power constraints, we backtrack to the previous time interval. If we backtrack to the beginning of the schedule and still cannot find feasible modes, we attempt the next legal configuration and select modes for idle intervals again.

## 6. Experimental Results

### 6.1. SCHEDULING

We use the NASA/JPL Mars rover [1] to evaluate the effectiveness our power-aware scheduling and task motion techniques. We construct a system-level representation that includes the computational, mechanical and thermal subsystems. The timing constraints on the heaters can be modeled with utilization-based constraints. We also consider the dual energy sources: solar panel and non-rechargeable battery. We consider three scenarios with different solar

power output levels: 14.9W (noon time), 12W, and 9W (dusk). The min power constraints are set to the respective solar outputs, while the max power constraints are set to the solar power plus 10W, which is the maximum battery power rating.

Fig. 13 compares the results of four techniques by using the energy cost to non-rechargeable battery and the execution time as metrics:

- (0) the existing manual solution,
  - (I) power-aware scheduling,
  - (II) power-aware task motion without utilization-based constraints,
  - (III) power-aware task motion with utilization-based constraints.
- In scenario 1, since the power budget is sufficient, a fast schedule is computed by all schedulers. However, solution I is quite expensive in terms of energy cost; II is cheaper. III delivers same performance with lower energy cost than both I and II, and it would not have been possible without utilization-based constraints.
  - In scenario 2, solutions I and II produce the same solution that is slower than scenario 1 due to the limited power budget. Solution III produces a fast schedule at a higher energy cost than I and II, but still within the max power constraint. No one solution is strictly better than the other, and they represent different trade-off points.
  - In scenario 3, the low power budget forces full serialization, and there is only one feasible, slow solution.

Without our task motion technique that aggressively explores the design space, the designers had no alternative choices for different scenarios but over-constrained the existing design for the worst case. However, the existing low-power solution draws less costly energy from the battery than our solutions. To evaluate this trade-off between performance and energy cost, we apply our schedules to a mission where the available solar power varies over time (Fig. 14) in three scenarios.

Suppose the rover is traveling to a target location in a distance of 48 steps. The mission starts with maximum solar power at 14.9W (Scenario 1). Then, it drops to 12W (Scenario 2) after 10 minutes, and falls to 9W (Scenario 3) 10 minutes later. If the existing serial schedule is applied, the rover will spend 10 minutes evenly in all three scenarios at a fixed slow moving speed. This results in a long execution time and very high energy cost in Scenario 3. On the other hand, our technique can produce two schemes. Both schemes use more free solar energy to speed up in scenarios 1 and 2 so that they can finish

| Scenario | (0) JPL's Low-power (hand-craft) | (I) Power-aware             | (II) Power-aware + Task motion | (III) Power-aware + Task motion + Utilization constraint |
|----------|----------------------------------|-----------------------------|--------------------------------|--|
| 1        | $\tau = 75s$ $Ec = 0J$ ✓         | $\tau = 50s$ $Ec = 79.5J$ ✗ | $\tau = 50s$ $Ec = 16.5J$ ✗    | $\tau = 50s$ $Ec = 4.5J$ ✓                               |
| 2        | $\tau = 75s$ $Ec = 55J$ ✓        | $\tau = 60s$ $Ec = 147J$ ✓  | same as (I)                    | $\tau = 50s$ $Ec = 208J$ ✓                               |
| 3        | $\tau = 75s$ $Ec = 388J$ ✓       | same as (0)                 | same as (0)                    | same as (0)  |

✓ = keep ✗ = drop

Figure 13. Comparison in three scenarios.  $\tau$  = makespan,  $Ec$  = energy cost above free min-power.

| Time frame (s)     | Scenario | JPL (0-0-0)     |             |                 | Task motion A (III-I-0) |             |                 | Task Motion B (III-III-0) |             |                 |
|--------------------|----------|-----------------|-------------|-----------------|-------------------------|-------------|-----------------|---------------------------|-------------|-----------------|
|                    |          | Distance (step) | Time (s)    | Energy cost (J) | Distance (step)         | Time (s)    | Energy cost (J) | Distance (step)           | Time (s)    | Energy cost (J) |
| 0 - 599            | 1        | 16              | 600         | 0               | 24                      | 600         | 129             | 24                        | 600         | 129             |
| 600 - 1199         | 2        | 16              | 600         | 440             | 20                      | 600         | 1470            | 23                        | 600         | 2482            |
| 1200 -             | 3        | 16              | 600         | 3114            | 4                       | 150         | 776             | 1                         | 10          | 85              |
| <b>Total</b>       |          | <b>48</b>       | <b>1800</b> | <b>3554</b>     | <b>48</b>               | <b>1350</b> | <b>2375</b>     | <b>48</b>                 | <b>1210</b> | <b>2696</b>     |
| <b>Improvement</b> |          |                 |             |                 |                         | <b>33%</b>  | <b>33%</b>      |                           | <b>49%</b>  | <b>24%</b>      |

Figure 14. Comparison over three scenarios of a mission.

the mission earlier to avoid costly scenario 3. Schemes A and B differ only in scenario 2 where A uses solution I while B uses the faster but more expensive solution III. As a result, scheme A achieves 33% speedup and 33% energy saving; and scheme B even further speeds up by 49% with a 24% energy reduction. These two alternative designs with different energy/performance trade-offs are discovered by our power-aware scheduling and task motion technique. They cannot be extracted otherwise by the existing techniques.

## 6.2. MODE SELECTION

We apply our algorithm to an example based on the Mars rover. Its resources consist of a camera (CAM), scientific devices (SCI), a radio-frequency modem (RF), a microprocessor (PPC), a hazard detector (HAZ), driving motors (DRV) and steering motors (STR). CAM takes a picture, sends the picture data to PPC for processing. Then, the rover moves to another location to perform scientific experiments.

Each resource has its own working modes. The microprocessor can operate at a number of different clock rates (with the maximum speed at 500MHz) and can be set to doze, nap or sleep modes. The RF modem can be in receive only mode, transmit-and-receive mode and sleep modes. The other resources have only two modes, on and off.

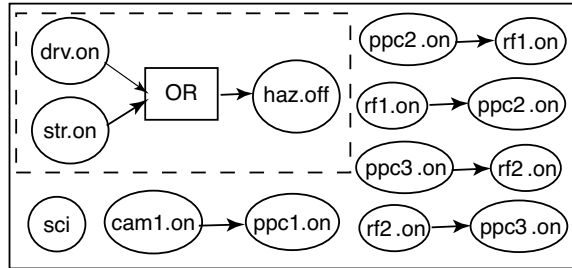


Figure 15. Mode Dependency Graph for the Mars Rover.

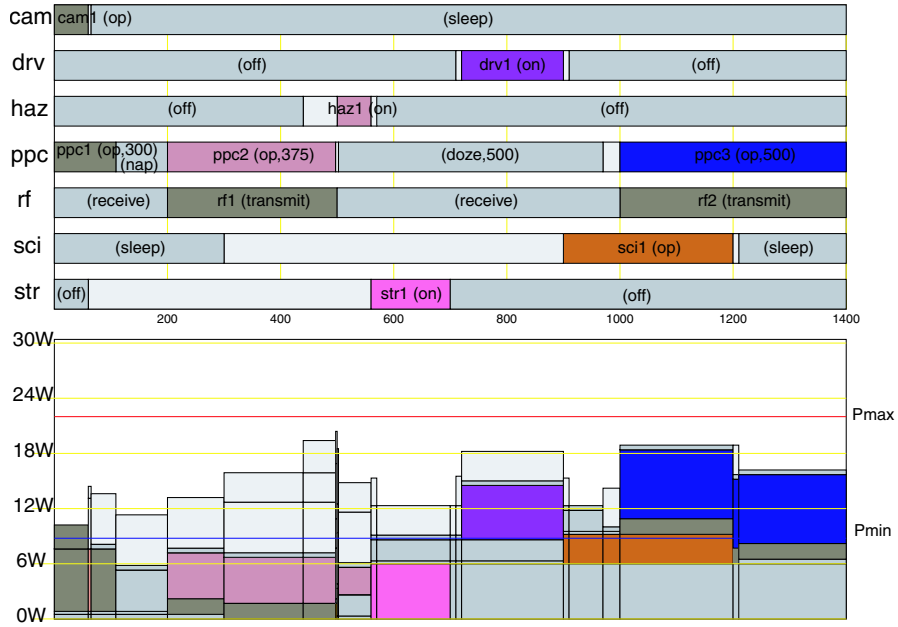
The mode dependency graph is shown in Fig. 15. For example, when hazard detector is working, neither the driving motor nor the steering motor should be turned on for power reasons. The RF modem is in transmit and receive mode if and only if the processor is processing data and communicating with the RF modem.

Fig. 16(a) shows a feasible mode schedule, in both the time view and power views. The tasks are labeled with task names and mode names. The dark gray areas are idle intervals labeled with mode names, while the light gray areas represent mode change overhead and are unlabeled.

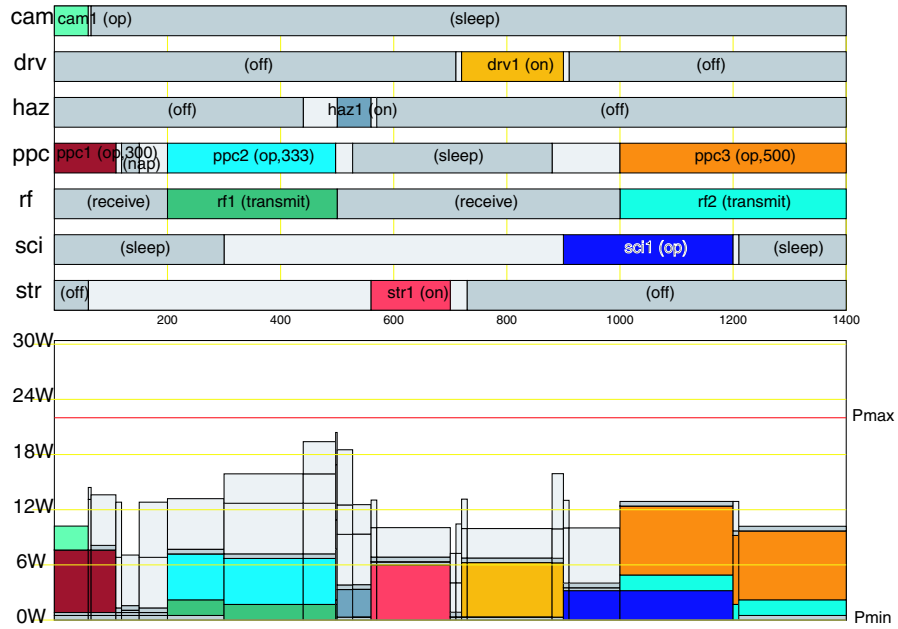
Task *ppc2* on the processor cannot be further slowed down because the processor and the RF modem must be co-active. If the processor is greedily slowed down, it will violate the max power constraint during the interval [500, 560] when the hazard detector is on. The idle interval between tasks *rf1* and *rf2* on the RF modem is set to receive only mode rather than off mode because the timing overhead of mode changes (power-down, power-up and pre-heating) is larger than the length of the interval. The idle interval before *rf1* is set to receive only mode for the same reason. The idle interval between tasks *ppc2* and *ppc3* on the microprocessor is set to doze,500 mode rather than sleep mode in order to meet the min-power constraint.

Figure 16(b) shows the result when the min-power constraint is set to zero. This allows the microprocessor to select sleep mode during the idle interval between tasks *ppc2* and *ppc3*, allowing other devices to select off mode during most of other idle intervals.

The availability of multiple modes of resources provides opportunities for finding better solutions. We compared our mode selection algorithm against two other approaches: (1) the *simple* approach that uses only two modes, on and off (for a processor, they are full-speed mode and off mode.); (2) the *greedy* approach that applies voltage scaling technique whenever possible. The results are shown in Fig. 17. Note that in all the three scenarios, the greedy approach violates the max power constraint. Our algorithm gives the best result because we use multiple modes of resources and apply voltage



(a)



(b)

Figure 16. (a) Mode schedule with max and min power constraints. The processor PPC cannot be greedily slowed down due to the max power constraint. (b) A mode schedule with max power constraint only. The result is similar to a low-power schedule.

| Scenario | Task sequence | $Cost_{simple}$ | $Cost_{greedy}$ | $Cost_{modesel}$ |
|----------|---------------|-----------------|-----------------|------------------|
| A        | CAM/MOV/SCI   | 19002           | 18442           | 17935            |
| B        | MOV/CAM/SCI   | 16381           | 15013           | 14667            |
| C        | CAM/SCI/MOV   | 20294           | 19505           | 19014            |

Figure 17. Comparison among different working scenarios. The tasks are CAM: shoot pictures; MOV: walk to another location; SCI: perform scientific experiments. (1) the simple approach assumes on/off modes; (2) greedily voltage scaling; (3) our mode selection algorithm, which always yields the best solutions.

scaling on the processor. At the same time, we avoid extra energy cost on the RF modem by identifying co-activation and performing mode selection.

## 7. Conclusions and Future Work

This paper presents the IMPACCT tool and methodology for system-level power management of power-aware embedded systems. The primary goal of the tool is to greatly expand the range of power/performance trade-offs, so that the system can most effectively adapt to the wide range of power availability in different operating scenarios. This can be accomplished by leveraging existing low-power and high-performance techniques, but a naive technique integration have led to incorrect results because important system-level properties were not properly considered. One of our contributions is precisely in modeling the important system-level dependencies including co-activation and inter-component modes. We have also developed power-aware scheduling and mode selection as two core tools for computing system-level power management policies. Our scheduler not only generates different schedules whose parallelism tracks the power availability, but also more aggressively increases the dynamic range by task motion while preserving timing and power constraints. Our mode selection methodology systematically exploits novel power management features in new components with a much richer set of power modes while considering all timing/power overhead associated with mode changes. All of these were made possible by our system-level dependency modeling methodology. Also supported is a system-level simulation engine that coordinates the execution of heterogeneous models that can range from native code to detailed simulation models and even emulators. They comprise a powerful framework to aid the quick exploration and validation of power management decisions. We believe this work represents a major step towards a framework that will be able to effectively integrate the best power management techniques developed by others and by us.

We are currently pursuing several directions for future work. One on-going project is to augment the library with a richer collection of components to include not only processor models but also more types of memory modules, peripheral devices, and battery models. To make our methodology practical and usable by engineers, we are also investigating automatic extraction techniques to reduce the effort in constructing the models needed as input to the IMPACCT tool. On scheduling, we are developing on-line, battery-aware algorithms under not only *power* constraints but also *energy* constraints. This will be supported by models for batteries and other energy sources [18, 29, 42]. Some initial work was recently proposed [24, 28], but we believe energy constraints must be considered in the context of the battery discharge and even recharge characteristics. Schedulers that are aware of battery discharge characteristics have been proposed [23, 29] but they do not treat power as constraints. On mode selection, it is being generalized to algorithm selection (e.g., between alternative image compression algorithms), which must be accompanied by data structure selection. Switching between algorithms and data structures will incur even larger timing and power overhead but the potential payoff will be tremendous. Together, we expect all these features will make IMPACCT a compelling tool for power-aware designs in the near future.

### Acknowledgements

This research is sponsored by DARPA under contract F33615-00-1-1719. It represents a collaboration between the University of California at Irvine and the NASA/Cal Tech Jet Propulsion Laboratory. Special thanks to Dr. N. Aranki, Dr. B. Toomarian, Dr. M. Mojarradi and Dr. J. U. Patel at JPL and Kerry Hill at AFRL for their discussion and assistance. The authors would also like to thank the anonymous reviewers for their very constructive feedback.

### References

1. 'NASA/JPL's Mars Pathfinder Home Page'. <http://mars3.jpl.nasa.gov/MPF/index0.html>.
2. Benini, L., A. Bogliolo, and G. D. Micheli: 2000, 'A survey of design techniques for system-level dynamic power management'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **8**(3), 299–316.
3. Benini, L., G. Paleologo, A. Bogliolo, and G. De Micheli: 1999, 'Policy optimization for dynamic power management'. *IEEE Transactions on Computer Aided Design* **18**, 813–833.
4. Chandrakasan, A., S. Sheng, and R. Brodersen: 1992, 'Low-power CMOS digital design'. *IEEE Journal of Solid-State Circuits* **27**(4), 473–484.

5. Chao, L.-F., A. LaPough, and E. H.-M. Sha: 1997, 'Rotation Scheduling: A Loop Pipelining Algorithm'. *IEEE Transactions on Computer Aided Design* **16**(3), 229–239.
6. Chung, E.-Y., L. Benini, and G. De Micheli: 1999, 'Dynamic Power Management Using Adaptive Learning Tree'. In: *Proc. International Conference on Computer-Aided Design*. pp. 274–279.
7. Dinçbas, M., H. Simonis, and P. V. Hentenryck: 1986, 'Solving a Cutting-Stock Problem in Constraint Logic Programming'. In: *Logic Programming: Proceedings of the 1988 Joint International Conference and Symposium*. pp. 42–58.
8. Heintze, N., S. Michaylov, and P. Stuckey: 1992, 'CLP(R) and some electrical engineering problems'. *Journal of Automated Reasoning* pp. 9:231–260.
9. Hines, K. and G. Borriello: 1997, 'Selective Focus as a Means of Improving Geographically Distributed Embedded System Co-simulation'. In: *Proc. International Workshop on Rapid System Prototyping*.
10. Hong, I., D. Kirovski, G. Qi, M. Potkonjak, and M. B. Srivastava: 1998, 'Power optimization of variable voltage core-based systems'. In: *Proc. Design Automation Conference*. pp. 176–181.
11. Hong, I., D. Kirovski, G. Qu, and M. Potkonjak: 1999, 'Power optimization of variable-voltage core-based systems'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **18**(12), 1702–1714.
12. Hwang, C.-H. and A. Wu: 1997, 'A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation'. In: *Proc. 1997 Design Automation Conference*.
13. Jacome, M., G. de Veciana, and C. Akturan: 1999, 'Resource constrained dataflow retiming heuristics for VLIW ASIPs'. In: *Proc. International Symposium on Hardware/Software Codesign*. pp. 12–16.
14. Jaffar, J., J.-L. Lassez, and M. Maher: 1986, 'A logic programming language scheme'. In: *Logic Programming: Relations, Functions and Equations*. pp. 441–468.
15. Lalgudi, K. and M. Papaefthymiou: 1996, 'Fixed-phase retiming for low power design'. In: *Proc. International Symposium on Low Power Electronics and Design*. pp. 259–264.
16. Leiserson, C. and J. Saxe: 1990, 'Retiming synchronous circuitry'. *Algorithmica* **6**(1), 5–35.
17. Li, D., P. H. Chou, and N. Bagherzadeh: 2002, 'Mode Selection and Mode-Dependency Modeling for Power-Aware Embedded Systems'. In: *Proc. Asian and South Pacific Design Automation Conference*.
18. Linden, H.: 1995, *Handbook of Batteries*. McGraw-Hill.
19. Liu, J., P. H. Chou, and N. Bagherzadeh: 2002, 'Power-Aware Task Motion for Enhancing Dynamic Range of Embedded Systems with Renewable Energy Sources'. In: *Proceedings of Second Workshop on Power-Aware Computing Systems*.
20. Liu, J., P. H. Chou, N. Bagherzadeh, and F. Kurdahi: 2001a, 'A constraint-based application model and scheduling techniques for Power-aware Systems'. In: *Proc. International Symposium on Hardware/Software Codesign*. pp. 153–158.
21. Liu, J., P. H. Chou, N. Bagherzadeh, and F. Kurdahi: 2001b, 'Power-Aware Scheduling under Timing Constraints for Mission-Critical Embedded Systems'. In: *Proc. Design Automation Conference*. pp. 840–845.
22. Luo, J. and N. K. Jha: 2000, 'Power conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems'. In: *Proc. International Conference on Computer-Aided Design*. pp. 357–364.
23. Luo, J. and N. K. Jha: 2001, 'Battery-aware Static Scheduling for Distributed Real-time Embedded Systems'. In: *Proc. Design Automation Conference*. pp. 444–449.
24. Ma, T.-L. and K. Shin: 2000, 'A user-customizable energy-adaptive combined static/dynamic scheduler for mobile applications'. In: *Proceedings 21st IEEE Real-Time Systems Symposium*. pp. 227–236.

25. Mozetic, I. and C. Holzbaur: 1991, 'Integrating numerical and qualitative models within constraint logic programming'. In: *Logic Programming: Proceedings of the 1991 International Symposium*. pp. 678–693.
26. Namgoong, W., M. Yu, and T. Meng: 1997, 'A high-efficiency variable-voltage cmos dynamic dc-dc switching regulator'. In: *IEEE International Solid-State Circuits Conference*. pp. 380–381.
27. Okuma, T., T. Ishihara, and H. Yasuura: 1999, 'Real-Time Task Scheduling for a Variable Voltage Processor'. In: *Proc. International Symposium on System Synthesis*. pp. 24–29.
28. Parikh, A., M. Kandemir, N. Vijaykrishnan, and M. Irwin: 2000, 'Energy-aware instruction scheduling'. In: *Proc. International Conference on High Performance Computing*. pp. 335–344.
29. Pedram, M., C.-Y. Tsui, and Q. Wu: 1999, 'An integrated battery-hardware model for portable electronics'. In: *Proc. Asian and South Pacific Design Automation Conference*. pp. 109–112.
30. Qiu, Q., Q. Wu, and M. Pedram: 1999, 'Stochastic Modeling of a Power-managed System Construction and Optimization'. In: *Proc. International Symposium on Low Power Electronics and Design*. pp. 194–199.
31. Qiu, Q., Q. Wu, and M. Pedram: 2000, 'Dynamic power management of complex systems using generalized stochastic Petri nets'. In: *Proc. Design Automation Conference*. pp. 352–356.
32. Quan, G. and X. S. Hu: 2001, 'Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage processors'. In: *Proc. Design Automation Conference*. pp. 828–833.
33. Sanchez, F. and J. Cortadella: 1995, 'Time-constrained loop pipelining'. In: *Proc. International Conference on Computer-Aided Design*. pp. 592–596.
34. Shin, Y. and K. Choi: 1999, 'Power conscious fixed priority scheduling for hard real-time systems'. In: *Proc. Design Automation Conference*. pp. 134–139.
35. Shin, Y., K. Choi, and T. Sakurai: 2000, 'Power optimization of real-time embedded systems on variable speed processors'. In: *Proc. International Conference on Computer-Aided Design*. pp. 365–368.
36. Simunic, T., L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli: 2001, 'Dynamic Voltage Scaling and Power Management for Portable Systems'. In: *Proc. Design Automation Conference*. pp. 524–529.
37. Simunic, T., L. Benini, and G. De Micheli: 1999, 'Event-Driven Power Management of Portable Systems'. In: *Proc. International Symposium on System Synthesis*. pp. 18–23.
38. Sinha, A. and A. Chandrakasan: 2001, 'Operating System and Algorithmic Techniques for Energy Scalable Wireless Sensor Networks'. In: *Proceedings of the 2nd International Conference on Mobile Data Management*.
39. Srivastava, M., A. Chandrakasan, and R. Brodersen: 1996a, 'Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation'. *IEEE Transactions on VLSI Systems* **4**(1), 42–55.
40. Srivastava, M., A. Chandrakasan, and R. Brodersen: 1996b, 'Predictive system shutdown and other architectural techniques for energy efficient programmable computation'. *IEEE Transactions on VLSI Systems* **4**(1), 42–55.
41. Stone, H.: 1996, 'Mars Pathfinder Microover: A Low-Cost, Low-Power Spacecraft'. In: *Proc. the 1996 AIAA Forum on Advanced Developments in Space Robotics*.
42. Wu, Q., Q. Qiu, and M. Pedram: 2000, 'An interleaved dual-battery power supply for battery-operated electronics'. In: *Proc. Asian and South Pacific Design Automation Conference*. pp. 387–390.
43. Yao, F., A. Demers, and S. Shenker: 1995, 'A scheduling model for reduced CPU energy'. pp. 374–382.

44. Yu, T. Z., F. Chen, and E. H.-M. Sha: 1998, 'Loop scheduling algorithms for power reduction'. In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*. pp. 3073–6.
45. Ziegenbein, D., K. Richter, R. Ernst, J. Teich, and L. Thiele: 1998, 'Representation of process mode correlation for scheduling'. In: *Proc. International Conference on Computer-Aided Design*. pp. 54–61.

*Address for Offprints:*

KLUWER ACADEMIC PUBLISHERS PrePress Department,  
P.O. Box 17, 3300 AA Dordrecht, The Netherlands  
e-mail: TEXHELP@WKAP.NL  
Fax: +31 78 6392500

