

IMPACCT: Methodology and Tools for Power-Aware Embedded Systems

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

Abstract

Power-aware systems are those that must exploit a wide range of power/performance tradeoffs 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 their 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.

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 tradeoffs. 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 tradeoffs 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.

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 [24].

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 pipelining, 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 pipelining 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 tradeoff 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 pipelining under pair-wise timing and total power as constraints. What distinguishes our work from traditional software pipelining and resource-constrained pipelining works is that we handle *co-activation* dependencies, 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 present a few simple examples to 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 [48], with follow-up by [30, 14, 31] 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, they typically slow down just enough without violating timing constraints, and many are based on real-time task scheduling cores [13, 38, 39, 36]. 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 [6]. Hong et al [13] proposed a heuristic for scheduling real-time tasks on a single variable voltage processor. Shin [38] exploited both execution time variation and idle time intervals for fix-priority tasks. Shin's algorithm in [39] determines the lowest maximum processor speed for each job to achieve power reduction. Quan and Hu [36] further greedily determine the lowest voltage for a set of tasks to achieve more energy savings.

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 [26, 28] presents 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 execution of one task requires the power consumption of other dependent services or tasks. 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 between sets of tasks.

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 it is not minimum energy. The max power constraint is violated because when task t_1 is stretched out, and when it overlaps task t_2

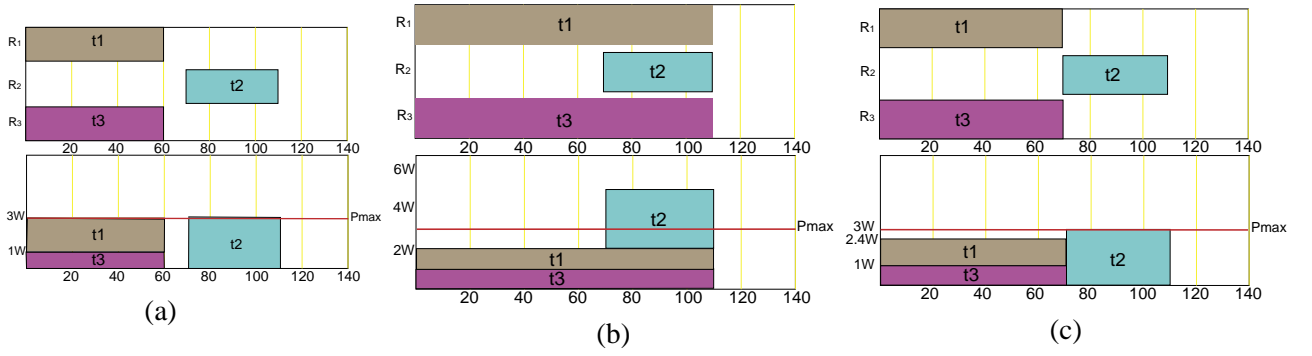


Figure 1: An application scenario that has resource dependency. (a) Initial schedule and power profile; (b) greedy voltage scaling results in a power spike that violate maximum power constraint; (c) a feasible solution meets both power and timing constraints, and saves energy as well.

Schedule	Timing violation	Power violation	Energy cost
Fig. 1(a)	No	No	300
Fig. 1(b)	No	Yes	320
Fig. 1(c)	No	No	288

Figure 2: Comparison of three schedules, greedy voltage scaling may violate the power constraint.

during the time interval from 70 and 110, their total power exceeds the max power constraint. It is not minimum energy 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, no max power is violated. Although task t_3 on resource R_3 is stretched with task t_1 and therefore consumes more energy than 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, overhead must be considered in determining the feasibility of the mode 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 times, adaptive timeout, or predictive based on a mix of profile and runtime history [44, 41, 10, 2]. The simplest power management policy is time-out based on a fixed or predicted amount of time before the system's shutdown or power-up [45, 16]. Stochastic models [3, 34] are used to address the uncertainty in system behaviors. Simunic et al [40] combines stochastic-modeled power management 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 [35], which models multiple service providers and their Generalized

	DPM		DVS		MS
	[45, 16]	[3, 34]	[13, 38, 39, 36]	[26, 28]	
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).

Stochastic Petri Net (GSPN) can capture some dependencies among resources. However, only one server is modeled to process an incoming request, and the GSPN model is mainly for the request/dispatch behavior of servers rather than dependency among the servers themselves. Without modeling this dependency, energy saved on the CPU may be more than offset by the increased energy consumed by the rest of the system.

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 entirely constraint driven, enabling us to make power/performance tradeoffs without hardwiring any specific goal or policy in the algorithm.

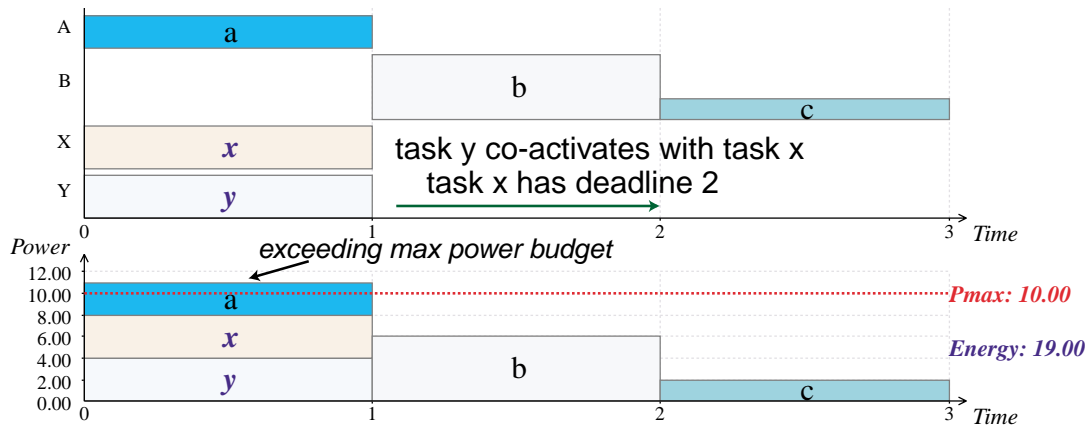
2.2 High Performance through Pipeline Transformation

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. first established the theoretical foundation for retiming synchronous circuits [20], and this has been extended to loop pipelining and scheduling for VLIW processors [37, 7, 17]. Shifting or “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 [18, 50].

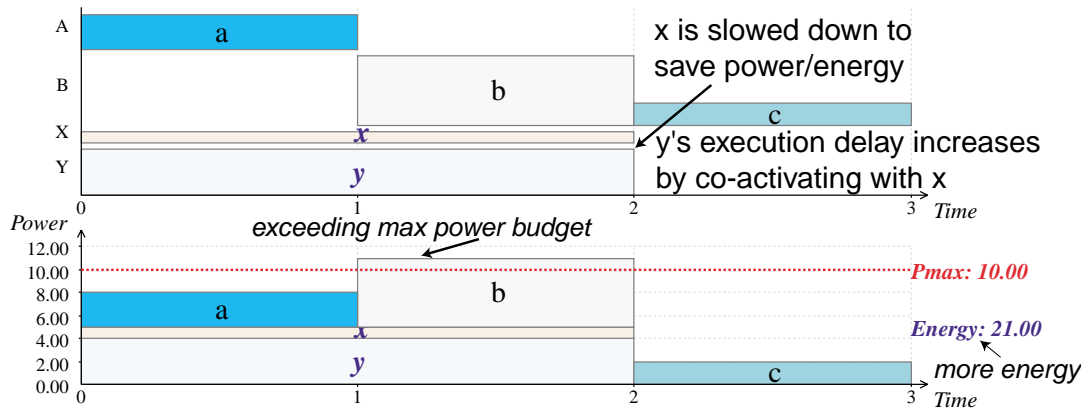
Existing techniques need significant enhancements before they can be correctly applied to our system-level power management problem. The main reason is that new types of dependencies must be modeled at the system level, and they cannot be readily handled by existing techniques by preprocessing. The tasks 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. We illustrate the problems with applying existing low-power and high-performance techniques to the system level.

Fig. 4 shows an example to illustrate limitation of some existing techniques compared to the optimal. It will be further used to explain our system model and scheduling algorithms in the ensuing text. In this example, five tasks a, b, c, x, y are to be scheduled on four execution resources A, B, X, Y . The constraints are:

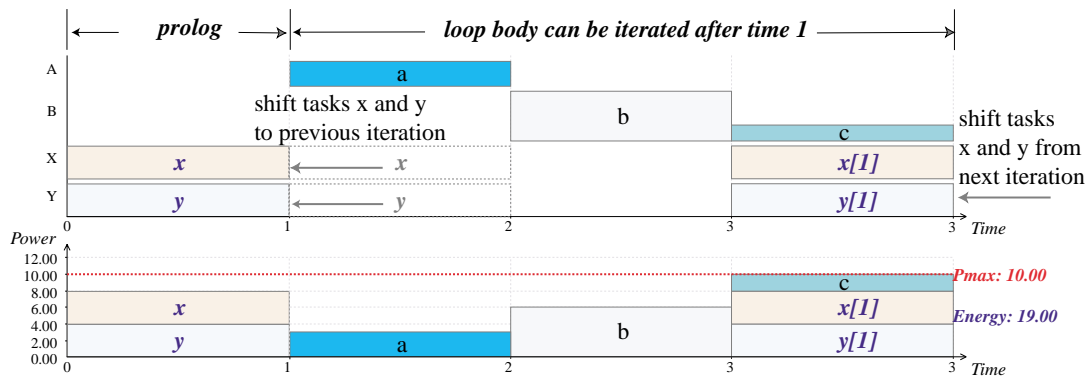
1. The overall deadline is at time 3.
2. The max power is 10.
3. The execution resources A, B are not voltage-scalable (e.g. they may be non-computation tasks).
4. Only task x can be voltage-scaled on resource X (e.g. a processor), and it has some slack time to finish before time 2.
5. Task y must be co-activate with task x , and its resource Y is also not voltage-scalable (e.g. memory, I/O).



(a) The schedule is not valid since max power budget is exceeded at time slot [0,1] due to parallel tasks x, y and a.



(b) DVS technique reduces power and energy consumption of task x. However, it fails to produce a valid schedule to the entire system. The energy consumption of the whole system is increased by co-activation.



(c) Our system-level loop pipelining technique shifts task x and its co-activated task y to the previous iteration such that the max power budget is satisfied.

Figure 4: An example where DVS fails to reduce power and energy at system level, while our technique will succeed

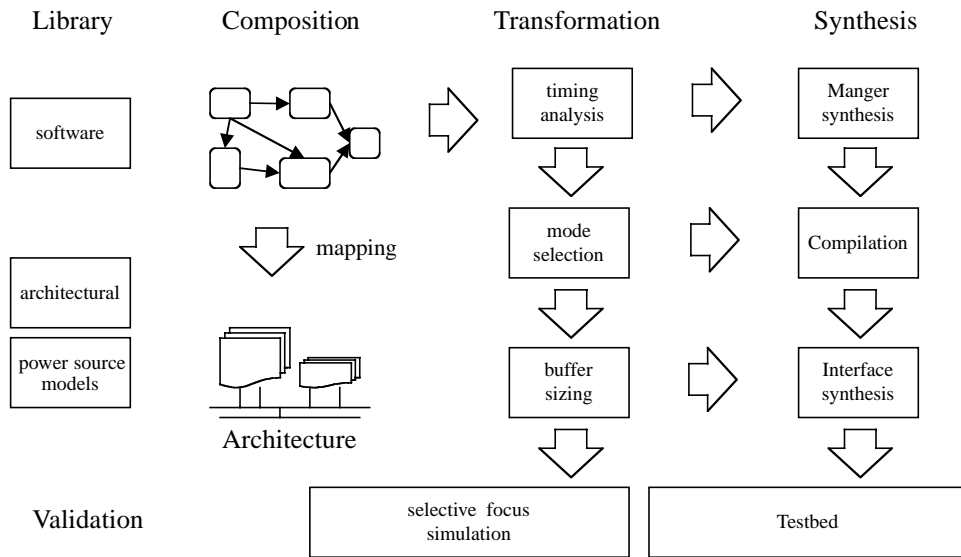


Figure 5: The IMPACCT system-level design tool for power-aware embedded systems.

Note that task y does not necessarily start and finish at the same time with x , but it has to start no later than x starts and finish no sooner than x finishes. For simplicity, in this example we assume x and y start and finish at the same time.

Fig. 4(a) shows a timing-valid schedule with a max-power violation during time $[0, 1]$. Rescheduling x and y in $[1, 2]$ will be timing-valid but still violating max power. Fig. 4(b) shows that if DVS was used to slow down task x until its deadline of time 2, we cut its energy by half according to CMOS scaling, and intuitively this should eliminate the max power violation. However, it actually increases total energy at the system level, because not only is DVS limited to x on the processor only, its lengthened execution time forces its co-activated task y to consume power over a longer time. Thus, energy saved by slowing down x is more than offset by energy increased by y . This is another example where DVS should not be applied in isolation.

Fig. 4(c) shows a feasible solution obtained by system-level pipelining, assuming iterative tasks. Tasks x and y are shifted (or *rotated*) to the previous iteration to overlap task c instead of a or b . As a result, both the max power and the deadline are satisfied. However, the optimal solution cannot be obtained unless we exploit domain-specific knowledge about the task set to eliminate a precedence dependency, which cuts the cycle time down to 2 time units. The details of rotation under min/max timing and the use of *pseudo-iteration* constraints will be explained in later sections.

3 Overview of IMPACCT

IMPACCT is a system-level design tool for exploring power/performance tradeoffs 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 critical in defense applications. IMPACCT is the first tool to correctly address all of these system-level power management issues. Fig. 5 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

tradeoffs 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. As an integration description, it has ports and channels for data dependency, as well as 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 equally important constraint: it will be a way to force the system to maintain activity above a certain level. The min and max constraints together will be a way to explore power/performance tradeoffs 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. Furthermore, 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 and power.

3.2 Target Architecture and Mapping

Also input to IMPACCT are 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 that 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 I²C 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.

3.3 Power-Aware Scheduling

The power aware scheduler supports several classes of power aware scheduling to yield the widest possible dynamic range of power/performance tradeoffs. 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 tradeoffs. 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, for data regular applications where the computation can be cleanly decomposed into stages the IMPACCT scheduler can exploit system-level pipelining stages in conjunction with processor throttling, the IMPACCT scheduler can 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 the mode schedule for configuring 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 mode 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. Fig. 6 shows a screen shot of the current version

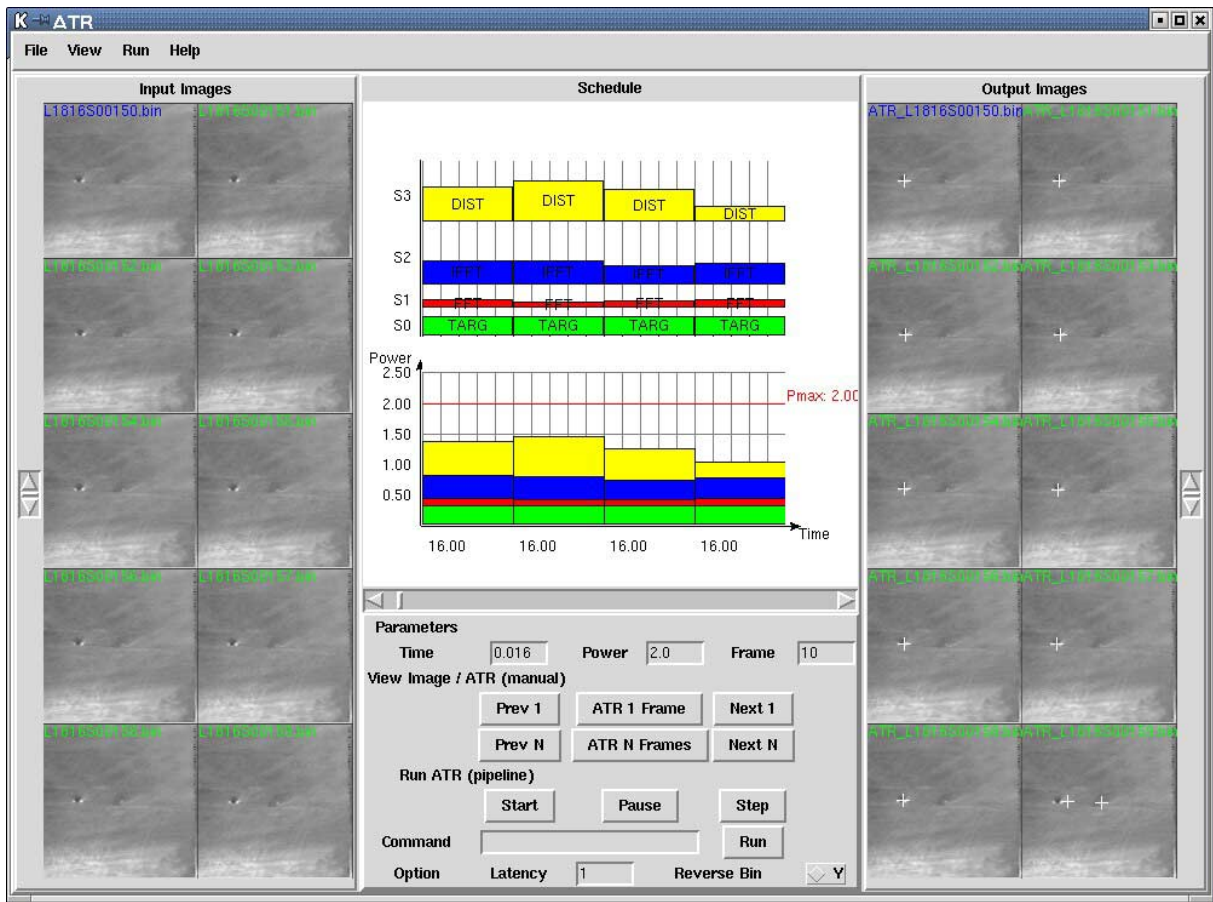


Figure 6: Screen shot of the IMPACCT graphical simulation front-end.

of the simulation tool running an automatic target detection (ATR) application. In the center panel is the power-aware Gantt chart showing the pipelined tasks and the simulation control panel. The left panel shows the input images, while the right panel shows the output images tagged with potential targets detected by the algorithm.

4 Power-Aware Scheduling

IMPACCT provides comprehensive scheduling support for maximal power/performance tradeoffs. We have developed static scheduling algorithms for satisfying timing and power constraints [?, 24], 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 pipeline transformation and aggressive mode selection. A number of rotation transformations 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 σ 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** σ assigns a start time $\sigma(v)$ to each task $v \in V$. It has a *finish time* τ_σ when all tasks complete their execution. Schedule σ 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, σ must also satisfy inter-iteration constraints such that they must hold across iterations when multiple copies of σ are concatenated.

A schedule σ 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 σ (illustrated by the power view of the Gantt-chart in Fig. 4). 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 σ 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 σ . 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.

Details of the scheduling algorithm can be found in [24]. Since this is an NP-hard problem, we have developed heuristics based on slacks to enable localized task movements, enabling the scheduler to quickly find feasible solutions without expensive backtracking.

4.2 Dynamic Range Enhancement through Pipelining

System-level pipelining can be an effective way to enhance the dynamic range of power/performance tradeoffs. We present a constraint model in a system-level context such that the component-level power management techniques can synergistically contribute to the improvement in both power and performance to the entire system. The distinguishing feature of our method from previous loop pipelining works is that, existing techniques either do not have timing constraints $\delta(u, v)$ in their data flow graphs (DFG), or the value of $\delta(u, v)$ is always 0 or 1 that only indicates precedence (data dependency). Moreover, we correctly model and handle *co-activation* dependency between components in a system. In addition, we propose a new class of timing constraints called *pseudo-iteration* constraints to enable more aggressive, domain-specific transformation for non-computational tasks.

It is performed in two steps: (a) transforming the problem into its pipelined versions, and (b) power-directed scheduling for each pipelined version. We first construct a timing constraint graph that expresses the tasks in an embedded system with pair-wise constraints. The pipelined version of a scheduling problem is obtained by rotation, when intra- and inter-iteration constraints are converted to each other. We first construct an **iteration graph** $G'(V, E')$: it has the same vertices as those of the constraint graph $G(V, E)$, but edges E' consist of only intra-iteration constraints. Formally, $E' = \{(u, v)\}, (u, v) \in E \text{ such that } \lambda(u, v) = 0, \delta'(u, v) = \delta(u, v)$. The expected loop duration τ is obtained from a non-pipelined schedule computed from the initial iteration graph G' .

Without loss of generality, we focus our discussion on down-rotations by which the execution of a task is shifted to the previous iteration of the loop, and the instance of the same task in the next iteration is included into the new loop body. The procedure for up-rotation can be similarly defined. A task v is *down-rotatable* if either vertex $v \in V$ does not have any incoming forward edges, or all of v 's incoming forward edges in G have at least one dependency depth. If σ is a valid non-pipelined schedule of one iteration, we can *down-rotate* a task v according to the *expected loop duration*, which is the finish time τ_σ of σ .

When a task v is rotated down by one iteration in graph G' , vertex v represents the execution of task v in the next iteration. Therefore, the new start time assignment $\sigma'(v) = \sigma(v) + \tau$. When a task v is being rotated, its corresponding min timing constraints (zero or positive values) will become max timing constraints (negative values), and its corresponding max timing constraints will transform into new min timing constraints.

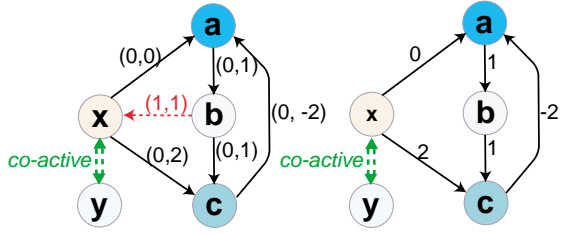
Fig. 7 illustrates the rotation of our example previously shown in Fig. 4. Each edge in the constraint graph G is denoted as (λ, δ) , while in iteration graph G' there is only δ . Inter-iteration constraints are marked as dashed arrows. Co-activation is denoted as a special pair of timing constraints. Fig. 7(a) shows the initial graphs G and G' and the schedule that violates the max power constraint. The non-pipelined schedule has a finish time $\tau = 3$. Fig. 7(b) shows rotation to task x and its co-activated task y to produce a new valid schedule (same as Fig. 4(b), except that the prolog is not shown) which otherwise cannot be achieved without rotation.

Task x can be rescheduled to time slot $[2, 3]$ because its outgoing min constraints are transformed into more relaxed max constraints ($\delta'(x, a) = -3, \delta'(x, c) = -1$, compared to 0 and 2 in Fig 7(a)). Tasks x and y are rotated together due to co-activation, but they are scheduled as separate tasks because they may not start and finish at the same time. Fig. 7(c) further rotates task a . The changes to edges in G' in Fig. 7(b) are reversed, and some new edges corresponding to a are changed. It gives a variation of the solution in Fig. 7(b). If tasks b and c are rotated next, the initial constraint graph in Fig. 7(a) will be restored.

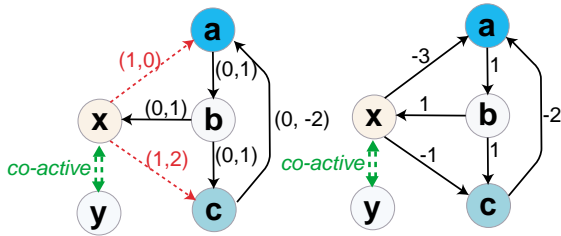
Rotation is based on the classification of inter-iteration and intra-iteration timing constraints. However, in some cases, it is difficult (or unnecessary) to decide whether a timing constraint should be inter-iteration or intra-iteration. Such cases are present in the Mars rover. For example, for timing constraints between a heater and a motor by which the motor is heated periodically, whether to model these constraints as intra-iteration or inter-iteration is not clear. In fact, whether the heaters and the motors stay in the same iteration does not matter. This is different from data dependencies in computation domain.

We define such type of constraints as *pseudo-iteration* timing constraints, which means the constraints can be expressed as either inter-iteration or intra-iteration. A pseudo-iteration 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 $\lambda(u, v) = *$,

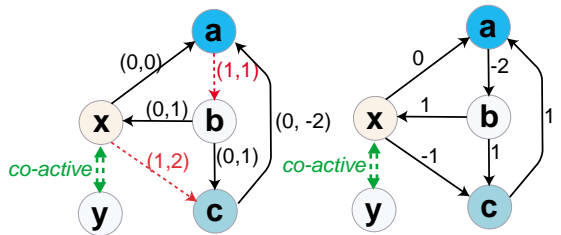
Constraint graph G Iteration graph G'



(a) before pipelining, no valid solution can be found.



(b) after rotating task x and its co-activating task y , a valid solution is found.



(c) after rotating task a , a variation of solution (b) is produced.

Schedule σ

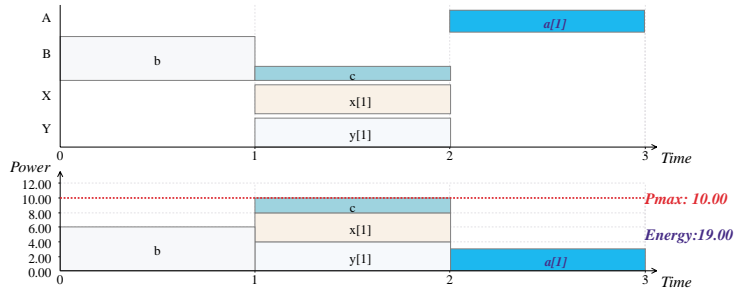
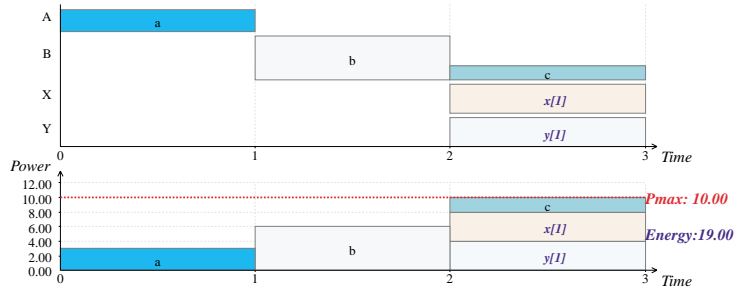
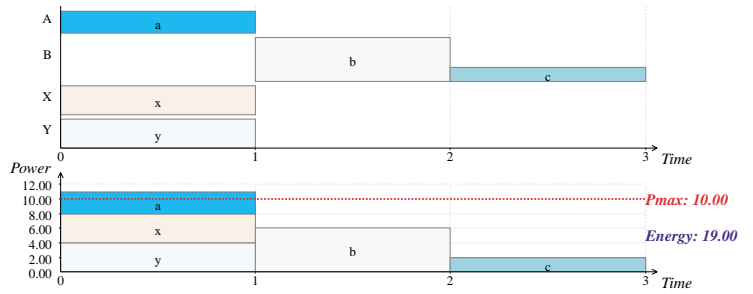
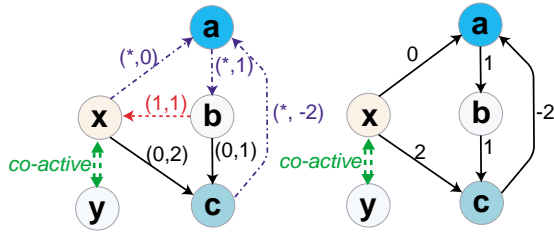
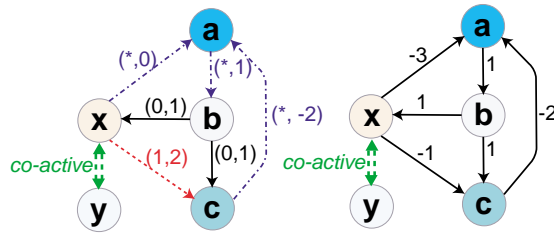


Figure 7: Rotation with timing constraints

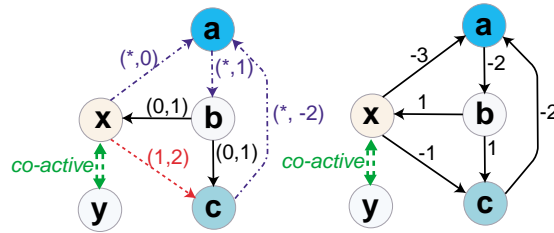
Constraint graph G Iteration graph G'



(a) before pipelining, no valid solution can be found.



(b) after rotating task x and its co-activating task y , a valid solution is found.



(c) after rotating task a with pseudo-iteration constraints, a new solution with better performance is found.

Schedule σ

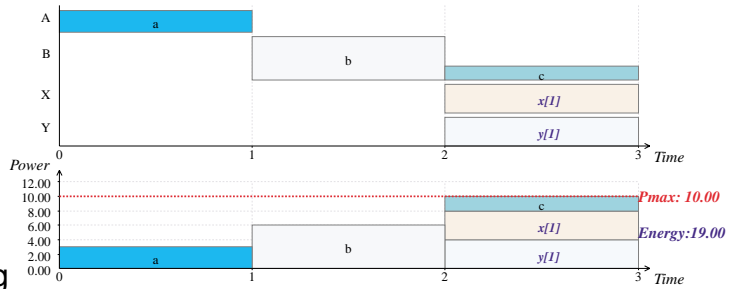
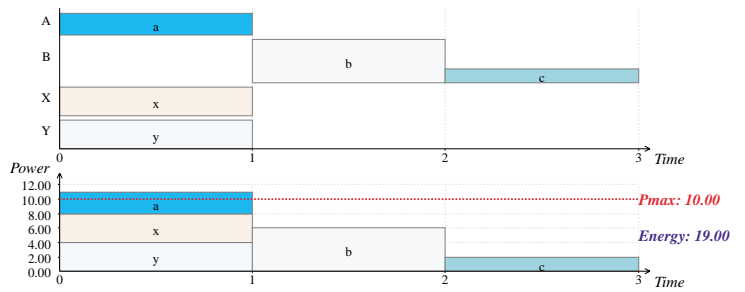


Figure 8: Rotation with pseudo-iteration constraints

indicating that it can be either zero or non-zero; and it will be included in iteration graph G' initially. Pseudo-iteration constraints will be always relaxed such that more scheduling opportunities become available.

For example, if resource A in our example is a heater and resource B are the motors, then the corresponding constraints of task a can all be modeled as pseudo-iteration ones. This is shown in Fig. 8(a) where pseudo-iteration constraints are marked as a different type of dashed arrows. Fig. 8(b) performs rotation to tasks x and y , similar to Fig. 7(b). In Fig. 8(c), when task a with pseudo-iteration constraints is rotated, the corresponding constraint values in graph G' are different from those in Fig. 7(c) in comparison. Specifically, we have the edge $\delta'(c, a) = -2$ as opposed to 1, and $\delta'(x, a) = -3$ instead of 0. Since the serialization chain formed by min constraints is broken, tasks a, b, c (after rotation to a , the chain becomes b, c, a in Fig. 7(c)) no longer have to be serialized. Now task a , a small power consumer, can overlap with b such that a surprising solution with shorter execution time ($\tau = 2$) is produced, and it also satisfies max power constraint. This optimal solution cannot be obtained without modeling pseudo-iteration constraints, which aggressively relax the constraints in the problem but are provably correct.

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) the modeling of 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) define the cost function based on the energy-delay product; and (4) efficiently generate feasible progression of these mode settings while satisfying all timing and power constraints.

5.1 Mode Attributes

Modes are attributes of a *resource*, which is a generalized term for a component in a system. A resource γ is defined as a graph $R_\gamma(M_\gamma, H_\gamma)$, where M_γ is a set of vertices, and $H_\gamma \subseteq M_\gamma \times M_\gamma$ is a set of edges. A vertex $m \in M_\gamma$ is a power mode of resource γ . An edge $(m, n) \in H_\gamma$ represents a mode change from mode m to mode n . we define the timing and energy function for a mode change as: $F : M_\gamma \times M_\gamma \rightarrow T \times En$, where M_γ is the set of modes of resource γ , and T, En are time and energy, respectively. The average power can be obtained from energy and time information.

For example, a processor may have **active**, **idle**, and **sleep** modes. Changing from any mode to other modes incurs time and energy overhead. Usually wakeup from **sleep** to **active** mode needs more time and energy than from **idle** to **active** mode. At architectural level, mode change overhead may be time and energy of changing hardware configurations only. At 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 like on, off, doze, nap, or sleep, but also encompass cache configurations for processors, different encoding/decoding techniques for radio antenna, variations of compiling techniques for software components, and different transmission protocols for bus drivers.

Each component is modeled as power and delay functions on modes and transitions. We define the mode set of a component, μ , which is the set of all the power mode $m \in \mu$. Power consumption is represented as a function, π , mapping from power mode to a power number. Formally, $\pi : \mu \rightarrow \mathbb{R}^+$. Delay of a mode transition is defined as a function, δ , mapping from start mode and end mode of a transition to a delay number. Formally, $\delta : \mu \times \mu \rightarrow \mathbb{R}^+$.

Given N resources $(\gamma_1, \gamma_2, \dots, \gamma_N)$, a mode combination $\pi \in \Pi$ of the resources (tasks) is a combination of N modes $\pi = (m_1, m_2, \dots, m_N)$. m_i is a mode assigned to resource γ_i (task τ_i), $1 \leq i \leq N$.

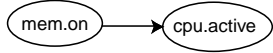


Figure 9: Mode dependency example: the memory is on only if the CPU is in active mode.

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 10: 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 output of the problem is an optimal configuration scheme with the minimum cost in terms of the power-delay-product. A configuration scheme is a table that maps each component to a globally optimal power mode over time.

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 co-activation, which forces another resource to select a given mode; it may also be exclusion, enabling, and many other possible types of dependency. These dependencies may be extracted from application level specifications or policies for safety, security, fault-tolerant, power-saving, or may be explicitly specified as mode correlation [51]. In any case, a *legal* mode combination of the resources is one that respects all of these dependencies, and a *feasible* mode combination 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 mode combinations in an order that facilitates the search for feasible combinations that are also low cost.

A mode dependency graph (MDG) $G_m(V, E)$ represents the inter-resource dependency relationships, where a vertex V is a resource mode, and a directed edge 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. 9). If the CPU is in sleep mode and the memory is on, then it violates the mode dependency. If the CPU is in sleep mode, and the memory is off, then it does not violate the mode dependency. Of course, if the CPU is in active mode and the memory is on, it satisfies the mode dependency. In addition, we also support logic 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.

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. 10. The meaning of AND, OR and XOR follows the normal boolean functions in the same names except when

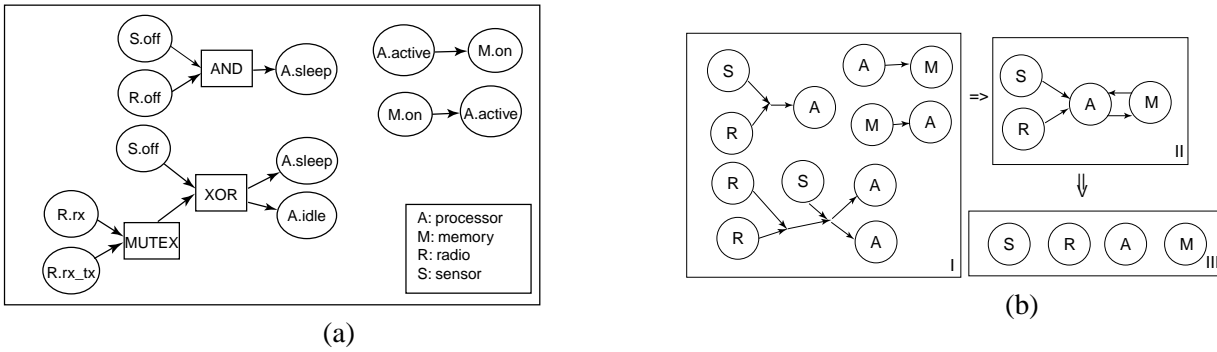


Figure 11: (a) An Mode Dependency Graph example: micro sensor. (b) Reduce an MDG to a resource list for generating mode combinations. Box I: from the MDG, shrink each operator vertex to a point, and remove mode name in each mode vertex. Box II: remove the redundant vertices and edges. Box III: break the cycle by removing one edge in the cycle, and apply topological sort to obtain a resource list.

any input is “undetermined,” the output is “undetermined.” When the operation of *MUTEX* (not listed in Fig. 10) is binary, its meaning is the same as *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. 11(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, it violates the dependency. If *S* is *on*, *R* is *off*, and *A* is in *sleep* mode, it does not violate the dependency.

Fig. 11(a) shows the mode dependency graph of the microsensor example[42]. 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 to represent the system cost for the schedule. By comparing the costs of different configurations, we determine an optimal solution for the problem. The cost function is in the form of power-delay product with two main parts: operating energy E_{op} and transition energy E_{tr} . We have $E_c = E_{op} + E_{tr} = P_{op}T_{op} + P_{tr}T_{tr}$. Operating energy is the energy consumed when the component is in a certain mode. Transition energy is the energy consumed when the component is switched from one configuration step to the next. Transition time is the delay on mode change. Sometimes either the transition power or transition time is large, which makes the transition cost high. We explore an optimal solution to have an configuration with minimum value of cost functions.

5.4 Mode Selection Algorithm

The mode dependency graph (MDG) captures enough information to enable the efficient generation of feasible mode combinations. The details of this algorithm is in [?], but basically it is a special version of topological traversal on the MDG with backtracking. In practice, however, backtracking is not necessary and the MDG has already pruned out all of the infeasible combinations. The search speed is much closer to linear than exponential. This section illustrates its application to a microsensor node in a distributed sensor network. It consists of a sensor, a processor, memory chips, radio frequency module and other auxiliary parts. When the sensor obtains information from environment, it sends data to the processor, the data is processed and sent to a basestation or other network node via RF module.

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

Figure 12: Modes of components in a microsensor.

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 13: Mode combinations generated from MDG.

The modes for the component is summarized in Fig. 12. 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 mode combinations for these components.

The inter-component relationships are specified using MDG as shown in Fig. 11(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 generate eight mode combinations that satisfy the given MDG (see Fig. 13). The generated mode combinations can be used by system power manager as legal modes in system level power management. This simple example demonstrated our algorithm’s ability to systematically generate legal mode combinations that satisfy inter-component dependencies, and by editing the mode dependency graph, we can obtain mode combinations without manually going through all possible mode combinations.

6 Experimental Results

6.1 Scheduling

We use the NASA/JPL Mars rover [1] to evaluate the effectiveness our system-level pipelining technique. We construct a system-level representation that includes the computational, mechanical and thermal subsystems. The timing constraints on the heaters can be modeled with pseudo-iteration 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.

Table 1 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) previous power-aware scheduling [24],
 - (II) system-level pipelining without pseudo-iteration constraints,
 - (III) system-level pipelining with pseudo-iteration 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 pseudo-iteration constraints.

Scenario	(0) JPL hand-craft	(I) P/A Non-pipelined	(II) Pipelined w/o pseudo-iter.cons.	(III) Pipelined w/ pseudo-iter.cons.
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

Table 1: Comparison in three scenarios

Time frame (s)	Scenario	JPL (0-0-0)			Pipeline A (III-I-0)			Pipeline 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
Total		48	1800	3554	48	1350	2375	48	1210	2696
Improvement						33%	33%		49%	24%

Table 2: Comparison in a comprehensive scenario

- 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 tradeoffs.
- In scenario 3, the low power budget forces full serialization, and there is only one feasible, slow solution.

Without our pipelining 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 tradeoff between performance and energy cost, we apply our schedules to a scenario where the available solar power varies over time (Table 2).

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 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 tradeoffs are discovered by our system-level pipelining 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 [46]. The rover travels on the surface of Mars to perform scientific experiments and shoot images. Its resources consists of a camera (CAM), scientific devices (SCI), a radio-frequency modem (RF), a microprocessor (PPC), a hazard detector (HAZ), driven motors (DRV) and steering motors (STR). CAM takes a picture, sends the picture data to PPC for processing, PPC outputs to RF, and then the rover moves to another location (HAZ, DRV, STR) to perform scientific experiments (SCI, PPC, RF).

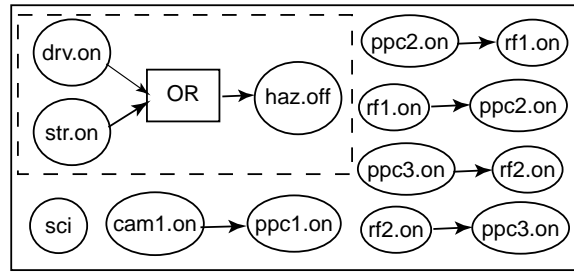


Figure 14: Mode Dependency Graph for the Mars Rover.

Each resource has its own working modes. The microprocessor can work at a number of different clock rates (with a full speed 500MHz) and can be set to **doze**, **nap** or **sleep** modes. 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**. Mode-change overhead is significant for some resources. The mode dependency graph is shown in Fig. 14.

For example, when hazard detector is working, neither the driving motor and steering motor should be working. 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. 15(a) shows a feasible mode schedule, in both the time view and power views. The tasks are labeled with task name and mode name. The gray areas are idle intervals labeled with mode names, while the light gray areas represent mode change overhead and are unlabeled.

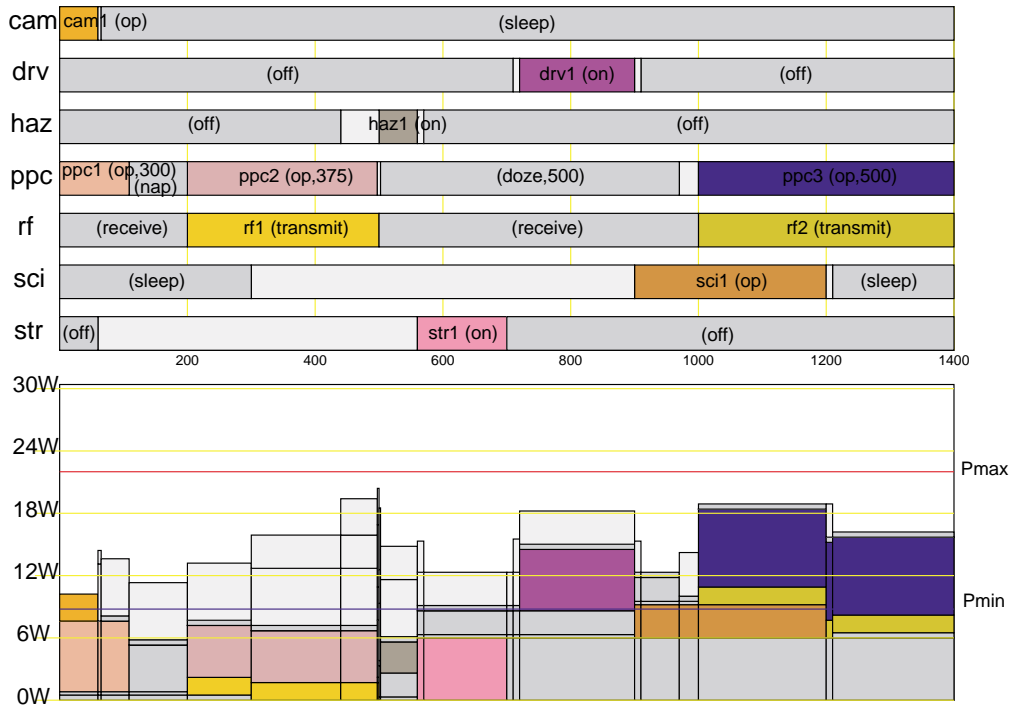
Task *ppc2* on the processor cannot be further slowed down because the processor and the RF modem must be co-active. If the processor is greedily slowed down, it will violate max power constraint during the interval 500 - 560 when the hazard detector is on. The tasks *drv1*, *haz1* and *str1* are not overlapped due to the system requirement specified by mode dependency graph. The steering motor and scientific device need significant time to pre-heat, which is adequately considered (the light gray areas in their tracks). Idle interval between task *rf1* and *rf2* on 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. 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 15(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 task *ppc2* and *ppc3*, allows other devices to select **off** mode during most of other idle intervals.

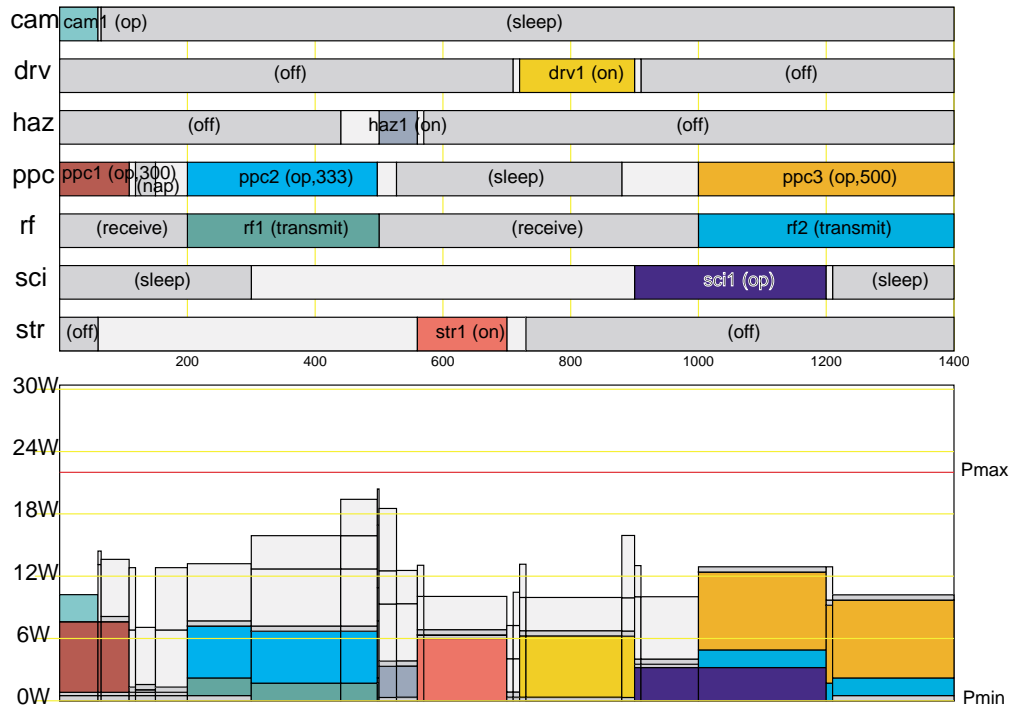
The availability of multiple modes of resources gives us opportunities to find better solutions. Our algorithm utilizes multiple modes and can effectively search mode schedules under system constraints. We compared our mode selection algorithm with two other approaches: approach one never utilizes multiple modes, but assumes only two modes, **on** and **off** (for a processor, they are **full-speed** mode and **off** mode.); approach two greedily applies voltage scaling technique whenever possible (we allow power constraint violation in this approach). The results are shown in Fig. 16. Note that in all the three scenarios, approach two violates max power constraint. Our algorithm gives best results because we utilized multiple modes of resources and apply voltage scaling on the processor. At the same time, we avoid extra energy cost on RF modem by identifying co-activation dependency between the two resources and performing mode selection to find the feasible solution.

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



(a)



(b)

Figure 15: (a) Mode schedule with maximum and minimum power constraints, the processor PPC cannot be greedily slow down due to the maximum power constraint. (b) A mode schedule with maximum power constraint only. The result is similar to a low power schedule.

Scenario	Task sequence	$Cost_{simple}$	$Cost_{greedy}$	$Cost_{modesel}$	percentage(simple/greedy/modesel)
A	CAM/MOV/SCI	19002	18442	17935	100% /97.0%/93.4%
B	MOV/CAM/SCI	16381	15013	14667	100% /91.6%/89.5%
C	CAM/SCI/MOV	20294	19505	19014	100%/96.1%/ 93.6%

Figure 16: Comparison among different working scenarios. Mission tasks: CAM: shoot pictures; MOV: walk to another location; SCI: perform scientific experiments. Approaches: simple: assume two modes; greedy: greedily voltage scaling; modesel: our mode selection algorithm.

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 intercomponent 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 pipeline transformation 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 ongoing 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. On scheduling, we are investigating 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 [21, 33, 47]. Some initial work was recently propose [29, 32], 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 [28, 33] 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. Finally, we are exploring acceleration of system-level power simulations based on the idea of *selective-focus* simulation [12]. The key idea is to dynamically switch the simulation resolution level for different components, depending on where detail is needed. All components outside the “focus” will be simulated at the highest level possible (fastest, least detail) to provide the workload to drive the other components. The focus will be selected based on the amount of power that the component draws as a percentage of the entire system. Together, we expect all these features will make IMPACCT a compelling tool for power-aware designs in the near future.

References

- [1] NASA/JPL’s Mars Pathfinder home page. <http://mars3.jpl.nasa.gov/MPF/index0.html>.
- [2] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, June 2000.
- [3] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. *IEEE Trans. Computer-Aided Design*, 18:813–833, June 1999.

- [4] D. Brooks, V. Tiwari, and M. Martonosi. Watch: a framework for architectural-level power analysis and optimizations. In *Proceedings of 27th International Symposium on Computer Architecture*, pages 83–94, June 2000.
- [5] D. Burger and T. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.
- [6] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.
- [7] L.-F. Chao, A. LaPough, and E. H.-M. Sha. Rotation scheduling: A loop pipelining algorithm. *IEEE TCAD*, 16(3):229–239, March 1997.
- [8] P. Chou, R. Ortega, and G. Borriello. Synthesis of the hardware/software interface in microcontroller-based systems. In *Proc. International Conference on Computer-Aided Design*, pages 488–495, 1992.
- [9] P. Chou, R. Ortega, and G. Borriello. Interface co-synthesis techniques for embedded systems. In *Proc. International Conference on Computer-Aided Design*, pages 280–287, 1995.
- [10] E.-Y. Chung, L. Benini, and G. De Micheli. Dynamic power management using adaptive learning tree. In *ICCAD*, pages 274–279, 1999.
- [11] Dalton Project. Basic introduction to the 8051 microcontroller. <http://www.cs.ucr.edu/dalton/i8051/>.
- [12] K. Hines and G. Borriello. Selective focus as a means of improving geographically distributed embedded system co-simulation. In *Proc. International Workshop on Rapid System Prototyping*, June 1997.
- [13] I. Hong, D. Kirovski, G. Qi, M. Potkonjak, and M. B. Srivastava. Power optimization of variable voltage core-based systems. In *Proceedings of Design Automation Conference*, pages 176–181, 1998.
- [14] I. Hong, D. Kirovski, G. Qu, and M. Potkonjak. Power optimization of variable-voltage core-based systems. *IEEE Trans. CAD of IC & Sys.*, 18(12):1702–1714, 1999.
- [15] C. Huang, B. Zhang, A.-C. Deng, and B. Swirski. The design and implementation of PowerMill. In *Proceedings. 1995 International Symposium on Low Power Design*, pages 105–108, April 1995.
- [16] C.-H. Hwang and A. Wu. A predictive system shutdown method for energy saving of event-driven computation. In *Proc. 1997 Design Automation Conference*, November 1997.
- [17] M. Jacome, G. de Veciana, and C. Akturan. Resource constrained dataflow retiming heuristics for vliw asips. In *Int. Symp. HW/SW Codesign*, pages 12–16, May 1999.
- [18] K. Lalgudi and M. Papaefthymiou. Fixed-phase retiming for low power design. In *ISLPEd*, pages 259–264, August 1996.
- [19] J. Larus. SPIM: A MIPS R2000/R3000 simulator. <http://www.cs.wisc.edu/larus/spim.html>.
- [20] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1990.
- [21] H. Linden. *Handbook of Batteries*. McGraw-Hill, 1995.
- [22] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi. A constraint-based application model and scheduling techniques for power-aware systems. In *Int. Symp. HW/SW Codesign*, pages 153–158, April 2001.
- [23] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *DAC*, pages 840–845, June 2001.
- [24] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *Proc. Design Automation Conference*, pages 840–845, June 2001.
- [25] J. Lorch and A. Smith. Software strategies for portable computer energy management. *IEEE Personal Communications*, (3):60–73, June 1998.
- [26] J. Luo and N. K. Jha. Power conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *Proc. Int. Conf. Computer-Aided Design*, pages 357–364, 2000.
- [27] J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *Proceedings of Design Automation Conference*, pages 444–449, 2001.

- [28] J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *Proc. Design Automation Conference*, pages 444–449, June 2001.
- [29] T.-L. Ma and K. Shin. A user-customizable energy-adaptive combined static/dynamic scheduler for mobile applications. In *Proceedings 21st IEEE Real-Time Systems Symposium*, pages 227–236, November 2000.
- [30] W. Namgoong, M. Yu, and T. Meng. A high-efficiency variable-voltage cmos dynamic dc-dc switching regulator. In *IEEE International Solid-State Circuits Conference*, pages 380–381, 1997.
- [31] T. Okuma, T. Ishihara, and H. Yasuura. Real-time task scheduling for a variable voltage processor. In *ISSS*, pages 24–29, November 1999.
- [32] A. Parikh, M. Kandemir, N. Vijaykrishnan, and M. Irwin. Energy-aware instruction scheduling. In *Proc. International Conference on High Performance Computing*, pages 335–344, December 2000.
- [33] M. Pedram, C.-Y. Tsui, and Q. Wu. An integrated battery-hardware model for portable electronics. In *Proceedings of the Asia and South Pacific Design Automation Conference 1999*, pages 109–112, January 1999.
- [34] Q. Qiu, Q. Wu, and M. Pedram. Stochastic modeling of a power-managed system construction and optimization. In *Proc. 1999 International Symposium on Low Power Electronics and Design*, pages 194–199, August 1999.
- [35] Q. Qiu, Q. Wu, and M. Pedram. Dynamic power management of complex systems using generalized stochastic petri nets. In *Proc. 2000 Design Automation Conference*, pages 352–356, 2000.
- [36] G. Quan and X. S. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of Design Automation Conference*, pages 828–833, 2001.
- [37] F. Sanchez and J. Cortadella. Time-constrained loop pipelining. In *ICCAD*, pages 592–596, November 1995.
- [38] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of Design Automation Conference*, pages 134–139, 1999.
- [39] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *International Conference on Computer-Aided Design*, pages 365–368, 2000.
- [40] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli. Dynamic voltage scaling and power management for portable systems. In *Proceedings of the Design Automation Conference*, pages 524–529, June 2001.
- [41] T. Simunic, L. Benini, and G. De Micheli. Event-driven power management of portable systems. In *ISSS*, pages 18–23, 1999.
- [42] A. Sinha and A. Chandrakasan. Operating system and algorithmic techniques for energy scalable wireless sensor networks. In *Proceedings of the 2nd International Conference on Mobile Data Management*, January 2001.
- [43] A. Sinha and A. P. Chandrakasan. JouleTrack – a web based tool for software energy profiling. In *Proc. Design Automation Conference*, pages 220–225, June 2001.
- [44] M. Srivastava, A. Chandrakasan, and R. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE TVLSI*, 4(1):42–55, March 1996.
- [45] M. Srivastava, A. Chandrakasan, and R. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Trans. VLSI Syst.*, 4(1):42–55, March 1996.
- [46] H. Stone. Mars pathfinder microrover: A low-cost, low-power spacecraft. In *Proc. the 1996 AIAA Forum on Advanced Developments in Space Robotics*, August 1996.
- [47] Q. Wu, Q. Qiu, and M. Pedram. An interleaved dual-battery power supply for battery-operated electronics. In *Proceedings ASP-DAC 2000. Asia and South Pacific Design Automation Conference 2000*, pages 387–390, January 2000.
- [48] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.
- [49] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. Irwin. The design and use of SimplePower: a cycle-accurate energy estimation tool. In *Proceedings 2000 Design Automation Conference*, pages 340–345, June 2000.

- [50] T. Z. Yu, F. Chen, and E. H.-M. Sha. Loop scheduling algorithms for power reduction. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3073–6, May 1998.
- [51] D. Ziegenbein, K. Richter, R. Ernst, J. Teich, and L. Thiele. Representation of process mode correlation for scheduling. In *Proceedings of International Conference on Computer Aided Design*, pages 54–61, November 1998.