Energy-aware implementation of hard-real-time systems upon multiprocessor platforms*

James H. Anderson Sanjoy K. Baruah

Abstract

Multiprocessor implementations of real-time systems tend to be more *energy-efficient* than uniprocessor implementations: since the power consumed by a CMOS processor is approximately proportional to the cube of the speed or computing capacity at which the processor executes, the total power consumed by an m-processor multiprocessor platform is approximately $(1/m^2)$ times the power consumed by a uniprocessor platform of the same computing capacity. However several factors, including the non-existence of optimal multiprocessor scheduling algorithms, combine to prevent all the computing capacity of a multiprocessor platform from being guaranteed available for executing the real-time workload. In this paper, this tradeoff — that *while increasing the number of processors results in lower energy consumption for a given computing capacity, the fraction of the capacity of a multiprocessor platform that is guaranteed available for executing real-time work decreases as the number of processors increases* — is explored in detail. Algorithms are presented for synthesizing multiprocessor implementations of hard-real-time systems comprised of independent periodic tasks in such a manner that the energy consumed by the synthesized system is minimized.

Keywords: Real-time systems; Energy-aware System Synthesis; Multiprocessor Scheduling Theory; Earliest Deadline First.

1 Introduction

With the proliferation of portable and mobile embedded systems, techniques for minimizing **energy consumption** are becoming increasingly important. Energy-aware system design and implementation is expected to become even more crucial as ubiquitous computing environments — in which relatively simple autonomous communicating devices (such as sensor networks) are distributed throughout a physical space — place an even greater premium upon battery life. The advent of these kinds of applications has led to a significant amount of research over the past decade on design methodologies and scheduling algorithms that consider the energy consumed by a system as a "first-class" concept, on par with logical and temporal correctness [8, 12, 17, 20, 22, 27, 28, 29, 31, 32, 33].

Multiprocessor real-time scheduling. Another topic of growing interest within the real-time systems community is **multiprocessor scheduling theory** [1, 3, 4, 6, 5, 7, 9, 11, 16, 18, 24, 25, 26]. This research has been motivated both by the advent of reasonably-priced multiprocessor systems, and also by a growth in computation-intensive real-time applications that have pushed beyond the capabilities of single-processor systems. In addition to offering more computing power,

*Supported in part by the National Science Foundation (Grant Nos. CCR-9988327, and ITR-0082866).

multiprocessor platforms often have the advantage of being more *energy efficient* than comparable uniprocessor platforms. As Wolf [30] has observed:

> The power consumption of a CMOS circuit is proportional to the square of the power supply voltage (V^2). Therefore, by reducing the power supply voltage to the lowest level that provides the required performance, we can significantly reduce power consumption. *We also may be able to add parallel hardware and even further reduce the power supply voltage while maintaining the required performance*. [Emphasis added.]
>
> — Wayne Wolf [30]

Simply put, energy consumption increases by a *multiplicative* factor as the speed of a single processor is increased, but by an *additive* factor as processors are added. This would suggest that it is advantageous to always use as many processors as possible. However, as processors are added, most real-time scheduling schemes suffer increasing schedulability loss, *i.e.*, their ability to make use of all available processing capacity declines. Consequently the pertinent question that arises when designing a particular application is: *what is the optimal number of processors to use if energy consumption is to be minimized?* This question is addressed in this paper.

Static versus dynamic voltage management. In non-real-time systems, the term *static power management* typically refers to mechanisms that are invoked by a user to manage power consumption (e.g., a "hibernate" mode invoked to save battery life while not using a laptop). In real-time and embedded systems, however, this term is more typically used to refer to energy-management schemes that are applied prior to run-time. In contrast, *dynamic power management* schemes (which include dynamic-voltage-scaling techniques) take actions to control power based upon the run-time activities of the CPU; for example, the CPU can automatically switch to a lower voltage (and hence execute at slower speed) if it is continually underutilized.

Depending upon such factors as the criticality of deadlines, the amount of computing overhead that can be devoted to power management, characteristics of the hardware being used, *etc.*, energy management may be performed at many levels:

System synthesis: Particularly for relatively simple, low-cost, low-power embedded systems that are to be mass-produced, the best (and perhaps only) time for performing energy optimizations may be during system design time. Energy-consumption requirements may greatly impact the choice of hardware components, and the scheduling and resource-allocation algorithms used. For mass-produced systems, it is worth putting considerable effort into the design process if even minor savings can be realized.

As a concrete example consider the Janus system [13, 14], a dual-core chip designed for controlling automotive power-train applications. This chip will eventually be deployed, *with an identical task workload*, in hundreds of thousands of automobiles. Consequently, clients are willing to use very computation-expensive (exponential-time) task-allocation algorithms to determine a good allocation once, and then hard-wire this allocation into all produced chips.

Static power management: Such schemes are particularly appropriate for embedded systems that are **(i)** heavily loaded at run-time (and hence require simple scheduling and energy-management schemes), or **(ii)** are extremely critical and thus need to be extensively tested to ensure predictability. Such systems may be scheduled with *table driven* and *time-triggered* [23] schedulers. In addition to providing scheduling information, lookup tables can store the voltage levels that should be supplied to various system components — under a static power management scheme, such voltages would be determined off-line.

Dynamic power management: If a system has the capability to dynamically adapt to changes in workload by changing energy consumption, then more sophisticated dynamic-voltage-scaling (DVS) techniques are possible. While DVS schemes may be able to save more energy, and work very well for soft- and non-real-time systems, they have some disadvantages in hard-real-time systems:

- *Complexity*: In most hard-real-time systems, there is a relationship between how *aggressive* the DVS algorithm can be (*i.e.*, how much energy it is able to save on average) and its computational complexity. DVS algorithms that are able to make real-time performance guarantees are typically quite complex. On very simple embedded systems (such as the Janus system [13, 14]), a complex power management scheme may itself place too heavy a load on the system.
- *Responsiveness*: The time taken for a processor to respond to changes in voltage supply can be on the order of tens or hundreds of milliseconds, particularly in "commodity" multi-level-voltage microprocessors. Hence, systems with widely varying real-time workloads comprised of jobs with tight deadlines may not allow sufficient time for switching a processor between power modes.

The discussion above suggests that there is a tradeoff between the complexity and sophistication of different energy management schemes, and their benefits in different scenarios. One of the long-term goals of our research is to identify the conditions under which each kind of energy management scheme may be most appropriate.

This paper. In this paper, we describe our research into the energy-efficient synthesis techniques for the construction of small embedded systems (such as the Janus automotive control system [13, 14]). Such systems typically have very well-defined workloads. We will assume that this workload is completely known at system design time, and is comprised either of **(i)** several independent jobs that all repeat with the same period,[1] or **(ii)** a collection of *sporadic tasks* (the sporadic task model is defined in Section 2). One of the major design goals in determining a scheduling strategy for such a system is *simplicity*: while it is acceptable to spend a considerable amount of time during (off-line) system synthesis, it should not be computationally expensive to make scheduling decisions at run-time. This requirement favors the use of simple on-line scheduling algorithms, such as table lookup schedulers (which are clock driven) or the earliest-deadline first scheduling algorithm (EDF) (which is event driven), which are known to have computationally efficient implementations. We impose the following requirements on our run-time system:

- First, we require that all the processors comprising the multiprocessor platform be provided the same supply voltage, resulting in them all being of equal speeds or computing capacities. Such multiprocessor platforms are referred to as **identical** multiprocessors.
- Next, we assume that our application is not amenable to table-driven scheduling. Due to additional constraints (such as those referred to above), it is necessary that we use an efficient run-time scheduling algorithm. We assume that this scheduling algorithm is EDF. One of

[1] Thus, the scheduling problem reduces to the problem of constructing a schedule of length equal to this common period, which is then repeatedly executed.

the advantages of EDF scheduling on identical multiprocessor platforms is that the number of preemptions and interprocessor migrations can bounded from above at the number of jobs being scheduled. (Upon multiprocessor systems in which different processors may have different speeds, implementations of EDF generally require a significantly larger number of preemptions and interprocessor migrations.)

Our approach in this paper is to first abstract out the "pure" scheduling-theoretic tradeoffs involved in choosing a particular multiprocessor computing problem — this gives rise to the **generalized multiprocessor feasibility (GMF) problem** (Definition 1). We motivate, formally state, and solve this scheduling problem in Sections 2.1 and 3. We believe that the GMF problem is interesting in its own right, and obtaining an efficient solution to it is one of the major contributions of this paper. In Section 4, we explain how solving the GMF problem permits us to synthesize energy-efficient multiprocessor real-time systems; in particular, we focus here on synthesizing EDF-scheduled systems upon multiprocessor platforms comprised of several identical processors.

Prior research on energy-aware multiprocessor scheduling. While a significant amount of research has been done over the last decade on addressing energy considerations in real-time systems (*e.g.*, see [12, 17, 21, 20, 27, 28, 31, 32, 33] — this list is by no means exhaustive), much of this work has focused on dynamic-voltage-scaling (DVS) techniques. Also, most previously-proposed algorithms are applicable either to uniprocessor systems, or to relatively simple table-driven, frame-based multiprocessor ones. Our research differs from most of the work cited above in that we are considering the issues of *system synthesis* and *pre-run-time scheduling* for significantly non-trivial multiprocessor systems. While DVS techniques could perhaps be used in conjunction with the techniques we devise, they are not directly related to our work and are in fact somewhat orthogonal to it.

Research is being conducted under several projects on the issue of energy-aware system synthesis, our focus here. Particularly notable are the **Power-aware Multiprocessor Architecture (PUMA)** project [22, 29] from the University of Southern California, and the MIT μ**AMPS** project [8]. Our research differs from these and other similar projects in that we are taking a more scheduling-theoretic approach: *we seek to adapt the most recent results in multiprocessor real-time scheduling theory for obtaining energy-aware system designs, and to help drive the agenda in multiprocessor scheduling research towards becoming more cognizant of energy-efficiency considerations.*

2 Background and motivation

Task and machine model. We assume that our multiprocessor platforms are preemptive shared-memory multiprocessors (SMP's), and that our scheduling model is as a consequence **preemptive** (i.e., a job executing on a processor may be interrupted and its execution resumed later at no cost or penalty), and allows for the interprocessor **migration** of jobs (a preempted job may resume execution upon the same or a different processor as the one it had executed upon prior to preemption, again at no additional cost).

As stated in Section 1 above, we will assume in this paper that our real-time workload is completely known at system design time, and is comprised of independent preemptive periodic tasks. First, we will consider the case where all these tasks have the same period. In this case, the problem of constructing an infinite schedule for the tasks reduces to the problem of determining a schedule for one period, which is then repeated over and over again. Hence, we will model our real-time workload as a collection of independent jobs, with the understanding that these jobs are repeated with a common period. (Such a strategy could of course also be used in case of differing periods, by constructing a schedule over the *hyperperiod* — the least common multiple of the individual periods — and repeating this schedule every hyperperiod. However, such an approach leads to combinatorial explosion in that exponentially many jobs of each periodic task will may need to be considered.)

A **hard-real-time job** $J = (a, c, d)$ is characterized by three parameters: an arrival time a, an execution requirement c, and a deadline d, with the interpretation that this job must receive c units of execution over the interval $[a, d)$. A hard-real-time **instance** I is comprised of a collection of such jobs.

In many embedded applications, the hard-real-time jobs comprising a real-time instance I are generated by a finite collection of sporadic tasks. A **sporadic task** $\tau_i = (C_i, T_i)$ is characterized by two parameters: an execution requirement C_i and a minimum inter-arrival separation parameter T_i. Such a sporadic task generates an infinite number of jobs, each having an execution requirement of C_i and a deadline T_i time units after its arrival time. The first job may arrive at any time-instant; successive arrivals are separated by at least T_i time units. A sporadic task system consists of several such sporadic tasks that are to execute on a specified processor architecture. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a sporadic task system. We will use the notation $I(\tau)$ to denote the real-time instance comprised of all the jobs generated by the sporadic tasks in sporadic task system τ.

2.1 Identifying the Design Space of Possible Multiprocessor Implementations

We have argued in the introduction that while it is generally more energy-efficient to use multiple processors instead of one, the total computing capacity needed to execute a particular workload tends to increase as the number of processors increases. Unfortunately, deducing the exact relationship between the number of processors required and the total computing capacity needed can be quite hard. We illustrate this point by considering the *generalized multiprocessor feasibility problem*, which we now define as a language-membership problem.

Definition 1 (The generalized multiprocessor feasibility (GMF) problem) Let $I = \{J_1, J_2, \ldots, J_n\}$ denote a hard-real-time instance, and S_{max} and S_{sum} be positive real numbers. We say that $(I, S_{\mathsf{max}}, S_{\mathsf{sum}}) \in \mathsf{GMF}$ iff instance I is feasible upon some platform of cumulative computing capacity S_{sum}, in which no processor has computing capacity greater than S_{max}.

Definition 2 Given a hard-real-time instance $I = \{J_1, J_2, \ldots, J_n\}$ and positive real number S_{max}, let $\mathsf{GMF}(I, S_{\mathsf{max}})$ denote the smallest value of S_{sum} such that $(I, S_{\mathsf{max}}, S_{\mathsf{sum}}) \in \mathsf{GMF}$.

■

How does the GMF problem relate to our problem of embedded-system synthesis to minimize energy consumption? For a given real-time workload I, if $\mathsf{GMF}(I, S_{\mathsf{max}})$ were to be independent of S_{max}, then as we have stated in the introduction (and will prove in Section 2.2), the most efficient implementation would be the one with maximum parallelism. However, we will illustrate via an example (Section 3) that $\mathsf{GMF}(I, S_{\mathsf{max}})$ is *not* independent of S_{max}; rather, it tends to *increase* as S_{max} decreases. Consequently, our synthesis problem is transformed to the problem of determining the *critical* value of the ordered pair $(S_{\mathsf{max}}, \mathsf{GMF}(I, S_{\mathsf{max}}))$ for which energy consumption is minimized.

As will be proved in Section 3, there is non-trivial relationship between the total computing capacity of any platform upon which a real-time instance is feasible, and the fastest processor in that platform: this is the relationship that is captured in the definition of the GMF problem. Determining the most-energy-efficient platform upon which workload I can be scheduled to meet all deadlines essentially requires that a *solution space* be determined, i.e., ordered pairs $(S_{\mathsf{max}}, S_{\mathsf{sum}})$ such that I can be scheduled to meet all deadlines upon a platform of cumulative computing capacity at most S_{sum} and in which no individual processor has computing capacity greater than S_{max}. For simple workloads comprised entirely of a fixed collection of independent jobs (as in the statement of the problem and the example above), we will show (Section 3) that this solution space can be represented as a collection of linear constraints, and that the GMF problem can be efficiently solved using standard linear programming techniques. However, the problem becomes considerably more difficult if the instance to be scheduled is represented more succinctly than by enumerating all jobs (e.g., as systems of periodic or sporadic tasks) or if resource-sharing constraints added. Solving this problem is an essential prerequisite to obtaining optimal or near-optimal energy-efficient implementations of embedded real-time systems.

2.2 How Multiprocessor Scheduling Relates to Energy Consumption

The effect of varying processor voltage (and hence varying the processor's energy consumption) upon the processor's computing capacity has been thoroughly explored during the last decade. The speed or computing capacity of a processor has been shown to be approximately directly related to its clock frequency f_c:

$$\mathsf{speed} \propto f_c \ .$$

For CMOS (and related) technologies [10], the Power consumed satisfies the following relationship:

$$\mathsf{Power} \propto f_c V^2 \ ,$$

where V denotes the voltage supplied to the processor. Operating at a lower supply voltage results in increased circuit delay and consequently decreased frequency f_c. The relationship here is [19]

$$f_c \propto \frac{(V - V_T)^2}{V} \ ,$$

where V_T denotes the threshold voltage and is a property of the processor chip. Hence, we see that

$$\mathsf{Power} \propto V(V - V_T)^2, \text{ and} \tag{1}$$

$$\mathsf{speed} \propto \frac{(V - V_T)^2}{V} \tag{2}$$

When $V_T \ll V$, we have the approximate relationships ($\mathsf{Power} \propto V^3$) and ($\mathsf{speed} \propto V$); i.e.,

$$\mathsf{Power} \propto \mathsf{speed}^3 \ . \tag{3}$$

This states that the power consumed by a processor is approximately proportional to the *cube* of its speed or computing capacity. That is, in obtaining a computing capacity of s on a uniprocessor platform, the power consumed is $k \times s^3$, where k is the constant of proportionality. If we were to instead obtain the same cumulative computing capacity by having m processors, each of computing capacity s/m, executing in parallel, then the total power consumption would be approximately

$$m \cdot k \cdot \left(\frac{s}{m}\right)^3 = \frac{1}{m^2} \cdot k \times s^2,$$

which is $\frac{1}{m^2}$ times the power consumed by a uniprocessor platform of equal capacity.

Observe that the power consumption of a multiprocessor platform relative to that of a comparable uniprocessor platform approaches zero as $m \to \infty$. This may suggest that we can reduce energy consumption to arbitrarily low levels by ***making the number of processors m as large as possible, with each processor of very low computing capacity***. However, this is clearly not possible: in the job model typically used in real-time scheduling (and assumed here), individual jobs are executed *sequentially*. Hence, there is a certain minimum speed that the processors must have if individual jobs are to complete by their deadlines: At the very least, computing capacity must be sufficient to complete each job between its arrival and its deadline.

In implementing energy-aware real-time systems on multiprocessor platforms, the tradeoff is thus as follows: *while increasing the number of processors results in lower energy consumption for a given computing capacity, the fraction of that capacity that is guaranteed available for executing the real-time workload decreases as the number of processors increases.*

3 The Generalized Multiprocessor Feasibility Problem

Recall the definition of the generalized multiprocessor feasibility problem:

Definition 1 (GMF problem (page 5)) For any hard-real-time instance I and positive numbers S_{max} and S_{sum}, $(I, S_{\mathsf{max}}, S_{\mathsf{sum}}) \in \mathsf{GMF}$ iff instance I is feasible upon some platform of cumulative computing capacity S_{sum}, in which no processor has computing capacity greater than S_{max}. ■

A note. A reasonable question to ask at this instant is this: *Why have we chosen to specify the* GMF *problem in terms of the total computing capacity and the computing capacity of only the* fastest *processor*? After all, the computing capacities of the processors other than the fastest also contribute to the total energy consumption: assuming that the constant of proportionality in Equation 3 is unity, the power consumption of a platform comprised of two processors of speeds one and one is $1^3 + 1^3 = 2$, while that of a platform comprised of three processors of speeds one, one-half, and one-half is $1^3 + (\frac{1}{2})^3 + (\frac{1}{2})^3 = 1.25$; this despite the fact that both platforms have the same cumulative capacities and fastest processors.

To answer this question, recall that while the GMF problem is concerned with the feasibility question ("does there exist a schedule of the real-time instance?") our goal is to implement real-time systems upon **EDF-scheduled identical multiprocessors**. To achieve this goal, we will make use of a result from [15] (reproduced below, as Theorem 3), which relates EDF-schedulability upon identical multiprocessors to feasibility upon a multiprocessor platform with known cumulative and maximum computing capacities. Since Theorem 3 makes use of only the cumulative capacity and the fastest processor speed, we have chosen to simplify the definition of the GMF problem by restricting its focus to just these two quantities.

In Example 1 below, we illustrate **(i)** how $\mathsf{GMF}(I, S_{\mathsf{max}})$ may be determined by careful case-analysis for particular values of I and S_{max}, and **(ii)** how solutions to the GMF problem may help determine energy-efficient multiprocessor implementations for I.

Example 1 Consider the real-time instance I comprised of the following three jobs:

$$J_1 = (0, 1, 1),$$

$$J_2 = (0, 2, 2), \text{ and}$$

$$J_3 = (0, 4, 4),$$

where (as before) $J = (a, c, d)$ denotes a job arriving at time instant a and needing to execute for c time units by a deadline at time instant d. Suppose that we wish to schedule this instance on a processing platform in which processor speeds will remain static during run-time (*i.e.*, *dynamic power management is not an option*).

Step §1: Solving the GMF. We consider four different scenarios:

Scenario 1: Suppose that the fastest available processor has a computing capacity of one unit of work per time unit. In that case, it is not difficult to see that each job must execute on a different processor and all processors must be of the same capacity. Such a platform has a cumulative computing capacity of three. Thus, $\mathsf{GMF}(I, 1) = 3$.

Scenario 2: Suppose that the fastest available processor has a computing capacity of (at least) 7/4 units of work per time unit. In that case, it may be verified that the EDF algorithm will successfully schedule all three jobs on a single processor of computing capacity 7/4, by executing J_1 over $[1, 4/7)$, J_2 over $[4/7, 12/7)$, and J_3 over $[12/7, 4)$. That is, the cumulative computing capacity of a platform comprised of individual processors of computing capacity no more than 7/4 units of work per time unit is 7/4. Equivalently, $\mathsf{GMF}(I, 7/4) = 1.75$

Scenario 3: Suppose that the fastest available processor has a computing capacity of 3/2 units of work per time unit. In that case, $\mathsf{GMF}(I, 3/2) = 2$; i.e., any platform that successfully schedules I must have a cumulative computing capacity of at least two. This is seen as follows:

- Over $[2, 4)$, job J_3 may execute on the faster processor, and thus complete $\frac{3}{2} \times 2 = 3$ units of work. This implies that a total of $1 + 2 + 1 = 4$ units of work ($e_1 + e_2 +$ the one unit of e_3 not completed during $[2, 4)$) must be done over $[0, 2)$. Hence, at least 2 units of work must be done in each time unit, *i.e.*, the cumulative computing capacity is at least two.
- The following schedule upon a platform comprised of two processors P_1 and P_2, of computing capacity 3/2 and 1/2, respectively, completes all three jobs by their deadlines:

  P_1: J_1 executes over $[0, \frac{2}{3})$, J_2 over $[\frac{2}{3}, 2)$, and J_3 over $[2, 4)$;
  P_2: J_3 executes over $[0, 2)$.

Scenario 4: Suppose that the fastest available processor has a computing capacity of 5/4 units of work per time unit. In that case, a 2-processor system with processor speeds equal to 1.25 and 1.0 respectively (i.e., $\mathsf{GMF}(I, 5/4) = 2.25$) will successfully schedule the system, as follows:

J_1 is scheduled during $[0, 1)$ upon the slower processor for one unit of execution,

J_2 is scheduled over $[0, 0.8)$ upon the faster processor and $[1, 2)$ upon the slower processor for a total of $(0.8 \times 1.25 + 1 \times 1) = 2$ units of execution, and

J_3 is scheduled over $[0.8, 4)$ upon the faster processor for a total of $(4.0 - 0.8) \times 1.25 = 4$ units of execution.

To see that no platform with fastest processor of speed 1.25 can have total capacity less than 2.25, notice that the work that must be done over the interval $[0, 2)$ is at least $(1 + 2 + 4) - 1.25 \times 2 = 4.5$; hence, at least 2.25 units of work must be performed per unit time.

Step §2: Choosing an energy-efficient design. Suppose that the available processors upon which to synthesize this system may run at a maximum speed of 2.0 units of execution per time unit. In Step §1 above, we explored the multiprocessor design space for this real-time system by iterating through the possible values of S_{max} starting at this maximum speed and iterating in steps of 0.25^2, and obtained the following possible configurations.

$\mathbf{S}_{\mathsf{max}}$	$\mathsf{GMF}(\mathbf{I}, \mathbf{S}_{\mathsf{max}})$	**configuration**
< 1.0	∞	(not feasible)
1.0	3.0	$[1, 1, 1]$
1.25	2.25	$[1.25, 1]$
1.5	2.0	$[1.5, 0.5]$
≥ 1.75	1.75	$[1.75]$

As we saw in Section 2.2 (Equation 3), the amount of energy consumed by a processor is approximately proportional to the cube of its speed. Assuming that the constant of proportionality in Equation 3 is equal to unity, the energy consumed by each of the four (feasible) configurations above would be $1^3 + 1^3 + 1^3 = 3$, $1.25^3 + 1.0^3 = 2.953125$, $1.5^3 + 0.5^3 = 3.5$, and $1.75^3 = 5.359375$, respectively. Consequently, the *second* configuration, with 2 processors of speeds of 1.25 and 1.0 respectively, would be optimal from the perspective of energy consumption. ■

Given a hard-real-time instance I and positive real numbers S_{max} and S_{sum}, we now describe how to construct a series of linear inequalities, denoted by (4)–(7) below. We prove (Lemmas 1 and 2) that these inequalities together have a feasible solution if and only if I is feasible upon a multiprocessor platform of cumulative computing capacity S_{sum} in which no processor has computing capacity greater than S_{max}. Determining whether I is feasible upon such a platform is thus transformed to the linear programming problem of determining whether the linear inequalities (4)–(7) have a feasible solution.

Observe that the $2n$ points $\bigcup_{i=1}^{n}\{a_i, d_i\}$ partition the time-line between $\min_{i=1}^{n}\{a_i\}$ and $\max_{i=1}^{n}\{d_i\}$ into p segments each of non-zero length, for some $p \leq 2n - 1$. Let us denote the length of the j'th such inteval as ℓ_j, for $1 \leq j \leq p$. We define $n \times p$ variables $x_{i,j}$, $1 \leq i \leq n$ and $1 \leq j \leq p$. The variable $x_{i,j}$ represents the rate at which job J_i executes during the j'th interval, for each $1 \leq i \leq n$ and $1 \leq j \leq p$.

- Certain of these variables are set equal to zero; specifically,

$$(\forall i : 1 \leq i \leq n : (\forall j : \text{ the } j\text{'th interval does not lie in } [a_i, d_i) \ : x_{i,j} = 0)) \ . \qquad (4)$$

  That is, for each i, the $x_{i,j}$'s corresponding to all intervals that lie before a_i or after d_i are set equal to zero.

- No job may execute at a greater rate during any interval than can be accommodated by the fastest processor available; this is formalized in the following constraints:

$$(\forall i : 1 \leq i \leq n : (\forall j : 1 \leq j \leq p : 0 \leq x_{i,j} \leq S_{\mathsf{max}})) \ . \qquad (5)$$

[2] In practice, we would iterate in much smaller increments — perhaps 0.01; for each value of S_{max} selected during this iteration, we would solve the corresponding GMF problem. Since this step is done once during system synthesis (as opposed to dynamically during run-time), the added computational cost of finer-grained iteration is not a major consideration.

- During each interval, the total amount of capacity assigned all the jobs cannot exceed the cumulative capacity of the entire platform; this is formalized in the following constraints:

$$(\forall j : 1 \leq j \leq p : \sum_{i=1}^{n} x_{i,j} \leq S_{\mathsf{sum}}) \ . \tag{6}$$

- For feasibility, each job must be allocated (at least) as much execution as its execution requirement; this is formalized in the following constraints:

$$(\forall i : 1 \leq i \leq n : \sum_{j=1}^{p} (x_{i,j} \times \ell_j) \geq e_i) \ . \tag{7}$$

The following two lemmas demonstrate the equivalence between the generalized multiprocessor feasibility problem and the problem of determining whether a collection of linear inequalities is feasible.

Lemma 1 Given a solution to the linear inequalities (4)–(7), we can construct a schedule meeting all deadlines for I upon any multiprocessor platform of cumulative computing capacity S_{sum} in which no processor has computing capacity greater than S_{max}.

Proof Sketch: Let $(x_{i,j})_{i=1,\ldots,n;j=1,\ldots,p}$ denote the solution to the linear inequalities (4)–(7). The schedule that assigns $(x_{i,j} \times \ell_j)$ units of execution to job J_i during the j'th interval for all $i, 1 \leq i \leq n$ and all $j, 1 \leq j \leq p$, would meet all deadlines of all jobs in I. ∎

Lemma 2 Given any schedule meeting all deadlines for I upon some multiprocessor platform of cumulative computing capacity S_{sum} in which no processor has computing capacity greater than S_{max}, we can obtain a feasible solution to the linear inequalities (4)–(7).

Proof Sketch: Let $(e_{i,j})_{i=1,\ldots,n;j=1,\ldots,p}$ denote the amount of execution assigned to job J_i during the j'th interval in the schedule meeting all deadlines for all $i, 1 \leq i \leq n$ and all $j, 1 \leq j \leq p$. A feasible solution to the linear inequalities (4)–(7) is obtained by assigning variable $x_{i,j}$ the value $(e_{i,j}/\ell_j)$, for all $i, 1 \leq i \leq n$ and all $j, 1 \leq j \leq p$. ∎

Observe that Constraints (4) and (5) together give rise to $n \times p$ linear inequalities, Constraints (6) give rise to p inequalities, and Constraints (7) give rise to n inequalities, for a total of $(np+p+n)$ inequalities. Since $p \leq 2n-1$, at most $(2n^2+2n-1)$ inequalities are generated — i.e., a *polynomial* number of inequalities. Now the problem of determining $\mathsf{GMF}(I, S_{\mathsf{max}})$ — the smallest S_{sum} such that hard-real-time instance I is feasible upon a platform with cumulative capacity S_{sum} in which no individual processor has computing capacity greater than S_{max} — is equivalent to solving the following linear programming problem:

Do the $(np+p+n)$ ***linear inequalities (4)–(7) have a solution?***

Thus, an instance of the GMF problem is converted, in polynomial time, into an instance of the linear programming problem on a polynomial number of variables. Since the linear programming problem is known to be solvable in polynomial time, it follows that

Theorem 1 The GMF problem has a polynomial-time algorithm.

∎

Sporadic task systems. If it is *a priori* known that $I = I(\tau)$ for some sporadic task system τ (i.e., that instance I is generated by sporadic task system τ), then the GMF problem is easily solved. Let

$$U_{\mathsf{sum}}(\tau) \stackrel{\text{def}}{=} \sum_{(C_i,T_i)\in\tau} \frac{C_i}{T_i}\,, \qquad \text{and} \qquad U_{\mathsf{max}}(\tau) \stackrel{\text{def}}{=} \max_{(C_i,T_i)\in\tau} \left\{\frac{C_i}{T_i}\right\}\,.$$

First, observe that we must have $S_{\mathsf{sum}} \geq U_{\mathsf{max}}(\tau)$, since it is possible that all tasks generate a job simultaneously and each task then generates subsequent jobs as soon as legal. Similarly, we must have $S_{\mathsf{max}} \geq U_{\mathsf{max}}(\tau)$, in order to be able to meet all deadlines of the maximum-utilization task under such a scenario. A platform comprised of n processors of computing capacities $u_1, u_2, \ldots, u_n$ respectively would have total computing capacity $U_{\mathsf{sum}}(\tau)$ and fastest processor computing capacity $U_{\mathsf{max}}(\tau)$, and hence be optimal in terms of simultaneously minimizing both S_{sum} and S_{max}. We therefore have the following result concerning the GMF when the workload is generated by a sporadic task system τ.

Theorem 2 Let τ denote a sporadic task system, and $I(\tau)$ range over all legal collections of real-time jobs generated by τ. Then

$$\mathsf{GMF}(I(\tau), S_{\mathsf{max}}) = \begin{cases} \infty & \text{if } S_{\mathsf{max}} < U_{\mathsf{max}}(\tau) \\ U_{\mathsf{sum}}(\tau), & \text{if } S_{\mathsf{max}} \geq U_{\mathsf{max}}(\tau) \end{cases} \tag{8}$$

■

That is, τ is not feasible upon a multiprocessor platform in which the fastest processor has speed less than $U_{\mathsf{max}}(\tau)$, regardless of the cumulative computing capacity of the platform; if processors with speed at least $U_{\mathsf{max}}(\tau)$ are available, then a cumulative computing capacity of at least $U_{\mathsf{sum}}(\tau)$ is (necessary and) sufficient to guarantee feasibility.

4 Energy-optimized system synthesis

§1. General workloads. For a given real-time instance I specified as a collection of independent jobs, we have seen (Example 1 in Section 3 above) how the space of possible multiprocessor designs can be explored by iterating through values of S_{max}; for each value of S_{max} so considered, the most energy-efficient implementation can be deduced (as was done in Example 1) and the implementation found to be most energy-efficient subsequently selected.

For real-time systems that are to be scheduled at run-time by table-driven schedulers, such an iterative approach to exploring the multiprocessor implementation space may suffice for obtaining the most energy-efficient system implementation (provided the algorithm used for solving the GMF problem during each iteration is *constructive*, *i.e.,* it either explicitly or implicitly constructs the required schedule). In many systems, however, a table-driven scheduler is not an option, either because the workload is dynamic, or because the table would be too large to store in memory. (For **recurring tasks**, a table of size proportional to the least common multiple of the task periods may be required. This value can be exponential with respect to the representation of the task system.)

Recall (from Section 1) that our objective is to design a method to synthesize an energy-efficient **EDF-scheduled identical multiprocessor implementation** of given real-time workload I. The GMF problem, on the other hand, is concerned with *feasibility* upon multiprocessor platforms in which all processors need not be of the same speed: for such platforms, the GMF problem restricts attention to the cumulative computing capacity of the platform the speed of the fastest processor.

To obtain an EDF-scheduled implementation upon identical multiprocessors from a solution to the GMF problem, we make use of the following result from [15], which relates feasibility upon (non-identical) multiprocessor platforms to EDF-feasibility upon identical multiprocessors.

Theorem 3 (Theorem 5 from [15]) Let I denote a hard-real time instance, which is feasible on a multiprocessor platform with total computing capacity S_{sum} in which the fastest processor has computing capacity S_{max}. Instance I is scheduled to always meet all deadlines on m processors each of computing capacity s by EDF, provided

$$S_{\mathsf{sum}} \leq m \cdot s - (m-1)S_{\mathsf{max}} \; . \tag{9}$$

■

Given hard-real-time instance I comprised of independent real-time jobs and positive real numbers S_{max} and S_{sum}, we had proved in Section 3 above (Lemmas 1 and 2) that the series of linear inequalities (4)–(7) defined in Section 3 together have a feasible solution if and only if I is feasible upon a multiprocessor platform of cumulative computing capacity S_{sum} in which no processor has computing capacity greater than S_{max}. Equivalently, any values for $(S_{\mathsf{max}}, S_{\mathsf{sum}})$ satisfying inequalities (4)–(7) represent a feasible implementation for I. Hence by Theorem 3 above, if $(S_{\mathsf{max}}, S_{\mathsf{sum}})$ satisfy inequalities (4)–(7) then I can be EDF-scheduled upon m identical processors of speed s each, where S_{max}, S_{sum}, m, and s satisfy Equation 9. And, the energy consumed by this implementation would be proportional to $m \times s^3$. Since our objective is to minimize the energy consumed, the systhesis problem can therefore be stated as the following (non-linear) optimization problem.

<u>**Problem *EnergyOpt***</u>

Minimize $\left(m \times s^3\right)$ subject to the following constraints:

1. m an integer;
2. $S_{\mathsf{sum}} \leq m \cdot s - (m-1)S_{\mathsf{max}}$; and
3. the linear inequalities (4)–(7).

Thisproblem can be solved using standard techniques for non-linear optimization.

§2. Sporadic workloads. For a real-time instance I that is *a priori* known to have been generated by a sporadic task τ system with known parameters, we have seen (Theorem 2) that there are unique optimal values for the speed of the fastest processor (S_{max}) and the cumulative computing capacity (S_{sum}) of multiprocessor platforms upon which I is guaranteed feasible: necessary and sufficient conditions for I to be feasible upon a multiprocessor platform are that the total computing capacity of the platform be at least $U_{\mathsf{sum}}(\tau)$, and the fastest processor be of speed at least $U_{\mathsf{max}}(\tau)$. When the optimal values of $(S_{\mathsf{max}}, S_{\mathsf{sum}})$ are unique (and can be easily identified) for a given real-time workload I in this manner, the energy-optimal EDF-scheduled identical multiprocessor implementation of I can be more efficiently determined than by solving the non-linear optimization problem above, as we now show. In the following, we set $S_{\mathsf{sum}} \leftarrow U_{\mathsf{sum}}(\tau)$, and $S_{\mathsf{max}} \leftarrow U_{\mathsf{max}}(\tau)$

From Equation 9, we can derive an expression for the minimum value of speed in terms of the number of processors m:

$$\mathsf{speed} \geq \frac{S_{\mathsf{sum}} - S_{\mathsf{max}}}{m} + S_{\mathsf{max}} \tag{10}$$

Hence, by Expression (10), the total power consumed when workload I is scheduled on m identical processors using EDF satisfies

$$\mathsf{Power}(m) \;\propto\; m \times \left(\frac{S_{\mathsf{sum}} - S_{\mathsf{max}}}{m} + S_{\mathsf{max}}\right)^3. \tag{11}$$

The value of m that minimizes the power consumed can be obtained by taking the derivative of the right-hand side of Expression (11) with respect to m, and setting the resulting expression equal to zero. Solving for m, we get

$$m = 2 \times \left(\frac{S_{\mathsf{sum}}}{S_{\mathsf{max}}} - 1\right). \tag{12}$$

Since the number of processors must be integral, we require that m be either the floor or the ceiling of the expression on the right-hand-side of Equation 12. To determine which one, we simply substitute both into the right-hand-side of Equation 3, which computes the total amount of energy consumed, and choose the value that yields the minimum energy consumption. In detail, this algorithm is

Algorithm *EnergyOpt-Sporadic*

1. Determine S_{max} and S_{sum}: by Theorem 2, $S_{\mathsf{max}} = U_{\mathsf{max}}(\tau)$ and $S_{\mathsf{sum}} = U_{\mathsf{sum}}(\tau)$.
2. Use Equation 12 to determine how many processors would be needed to obtain an energy-optimized implementation of I upon an EDF-scheduled identical multiprocessor platform. (Since Equation 12 may yield a non-integral result, both the ceiling and the floor of this result may need to be considered in the next step.)
3. Use Equation 3 to determine how much energy would be consumed if this particular EDF-scheduled identical multiprocessor configuration were to be used.

We illustrate by an example.

Example 2 Consider a sporadic task system τ with $U_{\mathsf{sum}}(\tau) = 2.1$ and $U_{\mathsf{max}}(\tau) = 0.8$. By Theorem 2, any instance $I(\tau)$ of jobs generated by this sporadic task system is feasible upon a multiprocessor platform with total computing capacity equal to 2.1 and in which the fastest processor has speed 0.8. By Equation 12,

$$m = 2 \times \left(\frac{2.1}{0.8} - 1\right) \quad = \quad 3.25$$

Since we cannot have a fractional number of processors, we must evaluate both the implementation with 3 processors, and the one with 4 processors:

With 3 **processors:** By Equation 10, the speed of each processor is $(2.1 - 0.83 + 0.8) = \frac{3.7}{3}$. By Equation 3, the power consumption of this implementation is $3 \times (\frac{3.7}{3})^3 = 5.62811$.

With 4 **processors:** By Equation 10, the speed of each processor is $(2.1 - 0.84 + 0.8) = 1.125$. By Equation 3, the power consumption of this implementation is $4 \times 1.125^3 = 5.6953125$

# processors m	$\mathsf{speed} = \frac{2.1-0.8}{m} + 0.8$ (by Equation 10)	$\mathsf{Power} = m \times \mathsf{speed}^3$ (by Equation 3)
1	2.1	$1 \times 2.1^3 = 9.261$
2	1.45	$2 \times 1.45^3 = 6.09725$
3	1.2333	$3 \times 1.2333^3 = \mathbf{5.62811}$
4	1.125	$4 \times 1.125^3 = 5.6953125$
5	1.06	$5 \times 1.06^3 = 5.95508$

Table 1: EDF-scheduled identical multiprocessor implementations of the sporadic task system of Example 2 on different numbers of processors.

Hence the most energy-efficient implementation of this hard-real-time task system as an EDF-scheduled identical-multiprocessor system is on 3 processors, each of computing capacity $\frac{3.7}{3}$. This is borne out by examining the implementations on one through five processors (Table 1).

■

§3. General workloads: When solving the general optimization problem is too expensive. While excellent tools exist for solving non-linear optimization problems such as the one that lies at the heart of Algorithm *EnergyOpt*, the fact remains that this problem may not have a polynomial-time solution. If the exponential time taken to solve this optimization problem is not acceptable, we can obtain a sub-optimal solution by applying an iterative technique (as we did in Example 1) to explore the design space of possible multiprocessor implementations of the system, by iterating through possible values of S_{max}. That is, we would iterate through possible values of S_{max} in steps of some predefined size (the finer the granularity of the iteration, the closer to the optimal solution we are likely to come). For the value of S_{max} considered during each iteration, we would determine $S_{\mathsf{sum}} \leftarrow \mathsf{GMF}(I, S_{\mathsf{max}})$, and use Steps 2 and 3 of Algorithm *EnergyOptSporadic* to compute the energy consumed if this particular configuration were to be implemented using EDF on identical multiprocessors. After all values for S_{max} have been iterated through, we would select for synthesis that particular configuration that yields the most energy-efficient implementation.

5 Conclusions

The advent of reasonably-priced shared-memory multiprocessor architectures, and corresponding advances in multiprocessor real-time scheduling theory, have made it possible to implement real-time systems upon multiprocessor platforms. One of the benefits of such multiprocessor implementations is energy efficiency: since the energy consumed by a CMOS circuit is approximately proportional to the square of the power supply voltage while the computing capacity is approximately proportional to the supply voltage, the total energy consumed by an m-processor multiprocessor platform is approximately $\frac{1}{m^2}$ times the power consumed by a uniprocessor platform of the same computing capacity. However, in contrast to the uniprocessor case in which provably optimal run-time scheduling algorithms (such as EDF) exist, not all the computing capacity of a multiprocessor platform may be available for executing a given real-time workload; in general, the fraction of the total computing capacity of an m-processor platform that is guaranteed available for executing a given arbitrary real-time workload tends to decrease with increasing m.

In this research, we have studied the problem of synthesizing a multiprocessor real-time system to implement a given real-time workload, with the objective of minimizing the energy consumed by the real-time system during run-time. Our target real-time applications are mass-produced embedded systems in which the real-time workloads are simply characterized and the run-time control system needs to be kept simple. Hence, we restricted our attention to

- *workloads* that are comprised either of **(i)** independent real-time jobs, each characterized by an arrival time, and execution requirement, and a deadline, such that all the jobs must be repeatedly executed with a common period (the "window" of the system), or **(ii)** independent sporadic tasks, each characterized by an execution requirement and a period (all periods are not required to be equal);
- multiprocessor platforms that are comprised of several *identical processors*, and that use *no dynamic energy management* techniques (such as Dynamic Voltage Scaling – DVS) during run-time; and
- systems that are scheduled using the widely-used, efficiently-implementable preemptive EDF *scheduling algorithm*.

For such systems, we have derived algorithms for optimally synthesizing multiprocessor implementations of given real-time workloads in order that the resulting system minimizes the amount of energy consumed. For workloads comprised entirely of sporadic tasks, we have shown (Algorithm *EnergyOpt-Sporadic*) that determining the energy-optimal implementation can be done in polynomial time. However for systems comprised of independent jobs that are repeatedly invoked with a common period, our algorithm (Algorithm *EnergyOpt*) for determining the energy-optimal implementation requires that a non-linear optimization be solved. Since the system is synthesized prior to run-time, we believe that this doesn not place an unreasonable computational requirement; however, if solving the non-linear optimization problem turns out to be computationally intractable, we have proposed (in Section 4) a polynomial-time technique for approximating the solution to the non-linear optimization problem.

References

[1] Abdelzaher, T., Andersson, B., Jonsson, J., Sharma, V., and Nguyen, M. The aperiodic multiprocessor utilization bound for liquid tasks. In *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium* (San Jose, California, September 2002), IEEE Computer Society Press.

[2] Anderson, J., and Baruah, S. Energy-aware implementation of hard-real-time systems upon multiprocessor platforms. Tech. Rep. TR02-045, Department of Computer Science, The University of North Carolina, 2002. Submitted for publication.

[3] Anderson, J., and Srinivasan, A. Early release fair scheduling. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Stockholm, Sweden, June 2000), IEEE Computer Society Press, pp. 35–43.

[4] Anderson, J., and Srinivasan, A. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Delft, The Netherlands, June 2001), IEEE Computer Society Press.

[5] Andersson, B., Baruah, S., and Jansson, J. Static-priority scheduling on multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 2001), IEEE Computer Society Press, pp. 193–202.

[6] Andersson, B., and Jonsson, J. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications* (Cheju Island, South Korea, December 2000), IEEE Computer Society Press, pp. 337–346.

[7] Baruah, S., Cohen, N., Plaxton, G., and Varvel, D. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica 15*, 6 (June 1996), 600–625.

[8] Bhardwaj, M., Min, R., and Chandrakasan, A. Power-aware systems. In *Proceedings of the 34th Asilomar Conference on Signals, Systems, and Computers* (Nov. 2000), vol. 2, pp. 1695–1701.

[9] Burchard, A., Liebeherr, J., Oh, Y., and Son, S. H. Assigning real-time tasks to homogeneous multiprocessor systems. *IEEE Transactions on Computers 44*, 12 (December 1995), 1429–1442.

[10] Chandrakasan, A. P., Sheng, S., and Brodersen, R. W. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits 27*, 4 (1992), 119–123.

[11] Davari, S., and Dhall, S. K. An on-line algorithm for real-time tasks allocation. In *Proceedings of the Real-Time Systems Symposium* (1986), pp. 194–200.

[12] Elnozahy, E., Melhem, R., and Mosse, D. Energy-efficient duplex and TMR real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium* (Austin, TX, December 2002), IEEE Computer Society Press.

[13] Ferrari, A., Garue, S., Peri, M., Pezzini, S., Valsecchi, L., Andretta, F., and Nesci, W. The design and implementation of a dual-core platform for power-train systems. In *Convergence 2000* (Detriot (MI), USA, October 2000).

[14] Gai, P., Lipari, G., and di Natale, M. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 2001), IEEE Computer Society Press.

[15] Goossens, J., Funk, S., and Baruah, S. Priority-driven scheduling of periodic task systems on multiprocessors. *Real Time Systems*. To appear.

[16] Ha, R., and Liu, J. W. S. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems* (Los Alamitos, June 1994), IEEE Computer Society Press.

[17] Havinga, P. J. M., and Smith, G. J. M. Design techniques for low-power systems. *Journal of Systems Architecture 46*, 1 (2000).

[18] Holman, P., and Anderson, J. Guaranteeing pfair supertasks by reweighting. In *Proceedings of the IEEE Real-Time Systems Symposium* (London, UK, December 2001), IEEE Computer Society Press.

[19] Hong, I., Kirovski, D., Qu, G., Potkonjak, M., and Srivastava, M. B. Power optimization of variable voltage core-based systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18*, 12 (1999), 1702–14.

[20] Hong, I., Potkonjak, M., and Srivastava, M. B. On-line scheduling of hard real-time tasks on a variable voltage processor. In *International Conference on Computer Aided Design (ICCAD-98)* (N. Y., Nov. 8–12 1998), ACM Press, pp. 653–656.

[21] Hong, I., Qu, G., Potkonjak, M., and Srivastava, M. Synthesis techniques for low-power hard real-time systems on variable voltage processor. In *Proceedings of the Real-Time Systems Symposium* (Madrid, Spain, December 1998), IEEE Computer Society Press, pp. 178–187.

[22] Kang, D., Crago, S., and Suh, J. Power-Aware design synthesis techniques for distributed Real-Time systems. In *Proceedings of the Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES-01)* (New York, June 22–23 2001), C. Norris and J. B. F. Jr., Eds., vol. 36, 8 of *ACM SIGPLAN Notices*, ACM Press, pp. 20–28.

[23] Kopetz, H., and Grünsteidl, G. TTP - A time-triggered protocol for fault-tolerant real-time systems. In *Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing (FTCS '93)* (Toulouse, France, June 1993), J.-C. Laprie, Ed., IEEE Computer Society Press, pp. 524–533.

[24] Liu, C. L. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary 37-60 II* (1969), 28–31.

[25] Lopez, J. M., Garcia, M., Diaz, J. L., and Garcia, D. F. Worst-case utilization bound for EDF scheduling in real-time multiprocessor systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Stockholm, Sweden, June 2000), IEEE Computer Society Press, pp. 25–34.

[26] Oh, D.-I., and Baker, T. P. Utilization bounds for N-processor rate monotone scheduling with static processor assignment. *Real-Time Systems: The International Journal of Time-Critical Computing 15* (1998), 183–192.

[27] Pillai, P., and Shin, K. G. Real-Time dynamic voltage scaling for Low-Power embedded operating systems. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP-01)* (New York, Oct. 21–24 2001), G. Ganger, Ed., vol. 35, 5 of *ACM SIGOPS Operating Systems Review*, ACM Press, pp. 89–102.

[28] Rusu, C., Melhem, R., and Mosse, D. Maximizing the system value while satisfying time and energy constraints. In *Proceedings of the IEEE Real-Time Systems Symposium* (Austin, TX, December 2002), IEEE Computer Society Press.

[29] Suh, J., Kang, D.-I., and Crago, S. Dynamic power management of multiprocessor systems. In *16th International Parallel and Distributed Processing Symposium* (Washington - Brussels - Tokyo, Apr. 2002), IEEE, p. 97.

[30] Wolfe, W. *Computers as Components: Principles of Embedded Computing Systems Design*. Morgan Kaufmann Publishers, 2000.

[31] Yao, F., Demers, A., and Shenker, S. A scheduling model for reduced CPU energy. In *36th Annual Symposium on Foundations of Computer Science: October 23–25, 1995, Milwaukee, Wisconsin* (1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1995), IEEE, Ed., IEEE Computer Society Press, pp. 374–382.

[32] Zhu, D., AbouGhazaleh, N., Mosse, D., and Melhem, R. Power aware scheduling for AND/OR graphs in multi-processor real-time systems. In *Proceedings of ICPP'2002, The 2002 International Conference on Parallel Processing* (Vancouver, B.C. Canada, August 2002).

[33] Zhu, D., Melhem, R., and Childers, B. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium* (London, UK, December 2001), IEEE Computer Society Press.