Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms*

James H. Anderson **Sanjoy K. Baruah**
The University of North Carolina at Chapel Hill

Abstract: Multiprocessor implementations of real-time systems tend to be more *energy-efficient* than uniprocessor implementations. 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.

1 Introduction

With the proliferation of portable and mobile embedded systems, techniques for minimizing energy consumption are becoming increasingly important. This has led to significant recent research 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 [14, 4, 8, 10, 11, 17].

Another topic of growing interest within the real-time systems community is *multiprocessor scheduling theory*. This research has been motivated both by the advent of reasonably-priced multiprocessor systems, and by a growth in computation-intensive real-time applications that have pushed beyond the capabilities of single-processor systems. Multiprocessor platforms often have the added advantage of being more *energy efficient* than comparable uniprocessor platforms. As Wolf [16] 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.]*

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 a pertinent question that arises when designing an energy-aware real-time application is: *what is the optimal number of processors to use if energy consumption is to be minimized?*

Static versus dynamic energy 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 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:

§1. 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

*Supported in part by the National Science Foundation (Grant Nos. CCR-9988327, CCR-0204312, and CCR-0309825).

even minor savings can be realized.

As a concrete example consider the Janus system [6, 7], a dual-core chip designed for controlling automotive power-train applications. This chip will eventually be deployed, *with an identical task workload*, in millions of automobiles.

§2. 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* [12] schedulers. In addition to providing scheduling information, lookup tables can store the (precomputed) voltage levels that should be supplied to various system components.

§3. 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. First, there is generally 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, and may themselves place too heavy a computational load on simple embedded systems (such as Janus). Furthermore, 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.

This paper. 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. In this paper, we describe our research into energy-efficient synthesis techniques for the construction of small embedded systems such as Janus [6, 7]. Such systems typically have very well-defined workloads. We will assume that this workload is completely known at system design time, and is comprised of a collection of periodic tasks (the periodic 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 the earliest-deadline first scheduling algorithm (EDF), which are known to have computationally efficient implementations. We impose the following requirements on our run-time system:

- We require that all the processors comprising the multiprocessor platform be provided the same supply voltage, resulting in them all being of equal computing capacities. Such multiprocessor platforms are referred to as **identical** multiprocessors.

- 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-like", in the sense that it is a *fixed-priority* algorithm (see Section 2.2). As we will see, one of the advantages of such fixed-priority 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 — hence our requirement above that all processors be of equal computing capacities.)

Prior research on energy-aware multiprocessor scheduling. Much prior work has focused on dynamic-voltage-scaling (DVS) techniques, and are typically applicable either to uniprocessor systems, or to relatively simple table-driven, frame-based multiprocessor ones. Our research differs from most of this prior work 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)* [11, 15] and the *μAMPS* [4] projects. 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 Model and Background

For our purposes, 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. We assume that our scheduling model is **preemptive** (i.e., an executing job may be interrupted and its execution resumed later at no cost or penalty), and may or may not allow for the interprocessor **migration** of jobs.

In many embedded applications, the jobs comprising a real-time instance I are generated by a finite collection of periodic tasks[1]. A **periodic 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 (often referred to as the *period* of the task). Such a periodic 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. We use the notation $U(\tau_i)$ to denote the *utilization* of task τ_i — $U(\tau_i) \stackrel{\text{def}}{=} C_i/T_i$. A periodic task system consists of several such periodic tasks. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a periodic task system. For any such periodic task system τ, $U_{\mathsf{sum}}(\tau)$ will denote the cumulative utilizations of all tasks in τ ($U_{\mathsf{sum}}(\tau) \stackrel{\text{def}}{=} \sum_{i=1}^{n} U(\tau_i)$), and $U_{\mathsf{max}}(\tau)$ will denote the largest utilization of any task in τ ($U_{\mathsf{max}}(\tau) \stackrel{\text{def}}{=} \max_{i=1}^{n} U(\tau_i)$). We will use the notation $I(\tau)$ to denote any real-time instance comprised of jobs generated by the periodic tasks in periodic task system τ.

2.1 How Multiprocessor Scheduling Relates to Energy Consumption

In recent research, the effect of varying processor voltage (and hence varying the processor's energy consumption) upon the processor's computing capacity has been thoroughly explored. 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 [5], the Power consumed satisfies the following relationship: $\left(\mathsf{Power} \propto V(V - V_T)^2\right)$, where V denotes the voltage supplied to the processor, and V_T denotes the threshold voltage and is a property of the processor chip. Operating at a lower supply voltage results in increased circuit delay and consequently decreased speed. The relationship here [9] is $\left(\mathsf{speed} \propto ((V - V_T)^2)/V\right)$.

[1]Note that what we call a periodic task here is sometimes referred to in the literature as a *sporadic* task.

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

$$\mathsf{Power} \propto \mathsf{speed}^3 \ . \tag{1}$$

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 $m \cdot k \cdot (s/m)^3 = \left(1/m^2\right) \cdot \left(k \times s^3\right)$.

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, as we will see in the following sections, it follows from real-time scheduling theory that the fraction of the computing capacity of a multiprocessor platform that is available for making real-time performance guarantees tends to *de*crease as the number of processors increases. 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.*

2.2 A classification of scheduling algorithms

Run-time scheduling is the process of determining, during the execution of a real-time application system, which job[s] should be executed at each instant in time. Run-time scheduling algorithms for identical multiprocessor platforms are often categorized along two orthogonal axes: the *priority-assignment* axis and the *inter-processor migration* axis.

§1: Priority assignment. Run-time scheduling algorithms are typically implemented as follows: at each time instant, assign a **priority** to each active job, and allocate the available processors to the highest-priority jobs. In **fixed-priority** scheduling, each job is assigned exactly *one* priority throughout its lifetime. (We distinguish between such fixed-priority algorithms and *static priority* algorithms for recurring real-time tasks — static priority scheduling algorithms are fixed-priority algorithms with the additional constraint that

all the jobs generated by each recurring task have the same priority.)

It can be shown that the total number of processor preemptions (and interprocessor migrations, if permitted) in a schedule generated by any fixed-priority algorithm is bounded from above by the total number of jobs being scheduled. Hence, the preemption and migration costs in fixed-priority scheduled systems can be amortized across all the jobs in the system, by simply inflating the execution requirement of each job by the amount of work needed to perform one preemption and one inter-processor migration — this is indeed a very important advantage of fixed-priority scheduling schemes over schemes that are not fixed-priority. In this paper, we are concerned with application systems in which it is important that the number of preemptions (and inter-processor migrations, if permitted) be bounded in this manner; hence in the remainder of this paper ***we restrict our attention to fixed-priority scheduling only***.

§2: Interprocessor migrations. Under **partitioned** scheduling, each task is assigned to a particular processor and all its jobs are only scheduled on that processor. Each processor schedules jobs independently from a local ready queue. Under **global** scheduling, by contrast, all ready jobs are stored in a single queue regardless of which job generated the task. A single system-wide priority space is assumed; the highest-priority job is selected to execute whenever the scheduler is invoked by any processor. Thus, there is no association between a job and a processor – a job that has been preempted upon a particular processor may later resume execution upon the same or a different processor.

It is known that the **utilization bound** of any partition-based algorithm – the largest utilization such that all periodic task systems with utilization no larger than this bound are guaranteed schedulable by that algorithm – cannot exceed $((\frac{m+1}{2}) \cdot \mathsf{speed})$ upon m processors each of computing capacity speed; if it is known that no individual task's utilization exceeds $U_{\mathsf{max}}(\tau)$, then a somewhat better bound of $((\frac{\beta m+1}{\beta+1}) \cdot \mathsf{speed})$ was proven by Lopez et al. [13], where $\beta = \lfloor \mathsf{speed}/U_{\mathsf{max}}(\tau) \rfloor$.

It has been shown [2] that (a variant of) EDF, called Algorithm fpEDF, can be implemented as a fixed-priority global scheduling algorithm to have a utilization bound that is about the same as the bound for partitioned EDF(see Figure 1):

$$\begin{array}{ll} m \cdot \mathsf{speed} - (m-1) \times U_{\mathsf{max}}(\tau), & \\ \qquad\qquad \text{if } U_{\mathsf{max}}(\tau) \leq \frac{1}{2} \times \mathsf{speed} & \\ \frac{m}{2} \times \mathsf{speed} + U_{\mathsf{max}}(\tau), & \\ \qquad\qquad \text{if } U_{\mathsf{max}}(\tau) \geq \frac{1}{2} \times \mathsf{speed} & \end{array} \tag{2}$$

(Global scheduling schemes such as the Pfair scheduling algorithms [3, 1] have been proposed that have a utilization bound of m upon m unit-capacity processors; hence, these scheduling algorithms are *optimal*. However, Pfair scheduling algorithms are not fixed-priority algorithms: each job generated by a periodic task may change its priority many times during its lifetime. One consequence of such priority-swapping is that Pfair-scheduled systems tend to have a large number of processor preemptions and interprocessor migrations. As stated above, we restrict our attention in this paper to the study of fixed-priority algorithms only.)

The utilization bounds, as a function of an upper bound U_{max} upon the largest individual utilization, obtainable by fixed-priority multiprocessor scheduling algorithms under the different migration restrictions discussed above, are plotted as a function of U_{max} in Figure 1; observe that the bounds for the restricted-migration and global cases are identical.

Algorithm fpEDF**.** We now briefly describe the global fixed-priority scheduling algorithm fpEDF. Basically, Algorithm fpEDF is a minor modification to global EDF — it assigns highest priority to jobs of tasks with large utilizations, and assigns priorities to jobs of other tasks as EDF would.

Suppose that task system τ is to be scheduled by Algorithm fpEDF upon m unit-capacity processors, and let $\{\tau_1, \tau_2, \ldots, \tau_n\}$ denote the tasks in τ indexed according to non-increasing utilization: $U(\tau_i) \geq U(\tau_{i+1})$ for all i, $1 \leq i < n$. Algorithm fpEDF first considers the $(m-1)$ "heaviest" (i.e., largest-utilization) tasks in τ. All the tasks from among these heaviest $(m-1)$ tasks that have utilization greater than one-half are treated specially in the sense that all their jobs are always assigned highest priority (note that this is implemented trivially in an EDF scheduler by setting the deadline parameters of these jobs to $-\infty$). The remaining tasks' jobs — i.e., the jobs of the tasks from among the heaviest $(m-1)$ with utilization $\leq$ one-half, as well as of the $(n-m+1)$ remaining tasks — are assigned priorities according to their deadlines (as in "regular" EDF).

3 Energy-optimized system synthesis

In this section, we present our algorithm for the energy-optimized synthesis of fixed-priority scheduled multiprocessor hard-real-time systems in which the workload is comprised of periodic tasks only, and in which all processors are required to execute at the same speed. Since the utilization bounds of partitioned scheduling is very close to that of global scheduling (except for the "step" nature of the bound in the partitioned case), we focus our attention on the utilization bound for global scheduling as given in Equation 2.

Suppose that we are given a collection of periodic

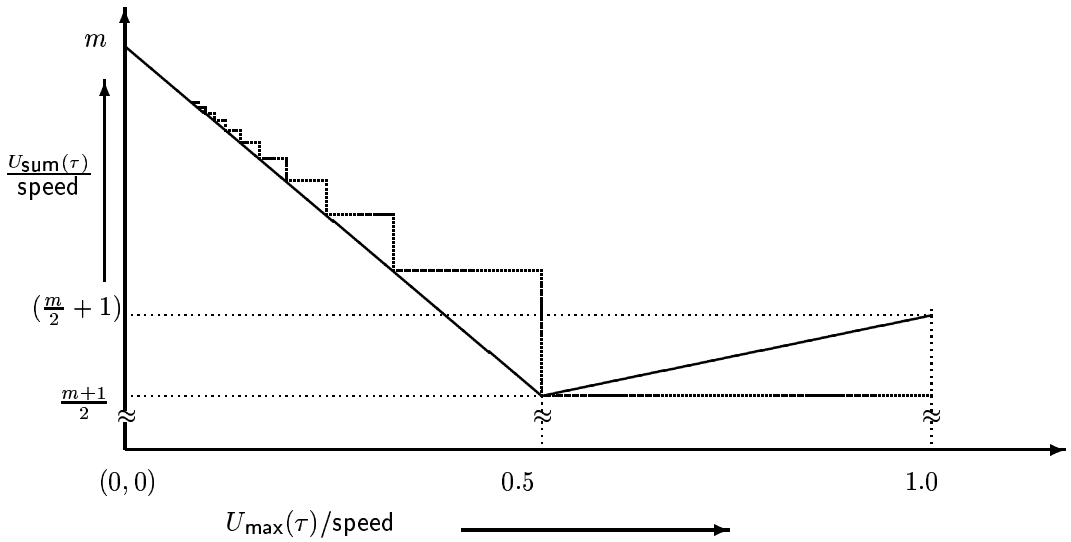

Figure 1. Utilization bounds for an m-processor system, with each processor having computing capacity speed. Utilization bounds are plotted on the y-axis, and the maximum utilization on the x-axis. For a given workload (i.e., a given τ), moving to a slower platform would move us further right on the x-axis. The solid line represents the bound for global scheduling; the dotted line, for partitioned scheduling. (The focus of this paper is primarily on the bound represented by the solid line.) The equation for the solid line is $y = m - (m-1)\,x$ for $x \leq \frac{1}{2}$, and $y = \frac{m}{2} + x$ for $x \geq \frac{1}{2}$ (explained in Section 2.2).

tasks τ, and need to implement τ on an identical multiprocessor system in the most energy-efficient manner possible. We consider two separate cases **(i)** when there is no *a priori* bound on number of processors that may be used, and **(ii)** when there is a limit on the number of processors that may be used. Our algorithm is given in Figure 2, and briefly summarized below; proofs follow in Sections 3.1 – 3.3.

(i) When the number of processors that may be used is not a constraining factor, the most energy-efficient implementation upon an identical multiprocessor platform has all processors executing at a speed that is equal to (or somewhat greater than) $U_{\mathsf{max}}(\tau)$. The optimal strategy would operate close to this point by choosing whichever of the following two configurations consumes less energy:

1. Either $m_1 \stackrel{\text{def}}{=} \lceil 2 \times (\Phi(\tau) - 1) \rceil$ processors at speed $U_{\mathsf{sum}}(\tau)$ each, where $\Phi(\tau) \stackrel{\text{def}}{=} U_{\mathsf{sum}}(\tau)/U_{\mathsf{max}}(\tau)$;
2. or $m_2 \stackrel{\text{def}}{=} \lfloor 2 \times (\Phi(\tau) - 1) \rfloor$ processors at speed $s(\tau, m_2)$ each (where $s(\tau, m_2)$ is computed as given by Equation 4 (Lemma 1).

(Note that if $\Phi(\tau)$ is large, so is the number of processors used. This agrees with our intuition that, if $U_{\mathsf{max}}(\tau)$ is very small when compared to $U_{\mathsf{sum}}(\tau)$, it is more energy-efficient to use a large number of processors.)

(ii) If other design considerations (such as limited availability of processors) prohibit the use of this many processors, then the best strategy is to use the largest number of processors permissible – let us denote this number as m_{max} – and execute each processor at a speed $s(\tau, m_{\mathsf{max}})$ (where $s(\tau, m_{\mathsf{max}})$ is again computed according to Equation 4 (Lemma 1).

We now prove these claims, in Sections 3.1 – 3.3 below.

3.1 Some preliminary results

Observe that an alternative way of writing the utilization bound function Equation 2 is as follows:

$$\max \quad \Big(m \cdot \mathsf{speed} - (m-1) \times U_{\mathsf{max}}(\tau) \;, \quad \frac{m}{2} \cdot \mathsf{speed} + U_{\mathsf{max}}(\tau) \Big) \tag{3}$$

since the "max" would be equal to the first term for $U_{\mathsf{max}}(\tau)/\mathsf{speed} < \frac{1}{2}$, and the second term for $U_{\mathsf{max}}(\tau)/\mathsf{speed} \geq \frac{1}{2}$ (see Figure 1).

Definition 1 *Let τ denote any periodic task system, and m any positive integer. The function $s(\tau, m)$ is defined as follows:*

$$s(\tau, m) \stackrel{def}{=} \textbf{min}\ s \ \textit{such that} \left[U_{\mathsf{sum}}(\tau) \geq \max \left(ms - (m-1)\, U_{\mathsf{max}}(\tau),\ \frac{m}{2} s + U_{\mathsf{max}}(\tau) \right) \right]$$

Input $U_{\mathsf{sum}}(\tau)$, $U_{\mathsf{max}}(\tau)$, and m_{max}, where
— τ denotes the periodic task system to be synthesized, and
— m_{max} denotes the maximum number of processors available

1. Let $\Phi(\tau) \stackrel{\text{def}}{=} \frac{U_{\mathsf{sum}}(\tau)}{U_{\mathsf{max}}(\tau)}$. Use Equation 1 to compute the energy consumed by each of the following two configurations:
   (a) $m_1 \stackrel{\text{def}}{=} \lceil 2 \times (\Phi(\tau) - 1) \rceil$ processors each operating at speed $U_{\mathsf{max}}(\tau)$; and
   (b) $m_1 \stackrel{\text{def}}{=} \lfloor 2 \times (\Phi(\tau) - 1) \rfloor$ processors each operating at a speed $s(\tau, m_2)$ as given by Equation 4 (Lemma 1).
2. If the lower-energy configuration from among these two uses $\leq m_{\mathsf{max}}$ processors, then this is the energy-optimal configuration; else, the energy-optimal configuration uses m_{max} processors each operating at a speed $s(\tau, m_{\mathsf{max}})$ given by Equation 4 (Lemma 1).

Figure 2. Algorithm for energy-optimal synthesis.

That is, $s(\tau, m)$ denotes the minimum speed *at which each processor in an m-processor identical multiprocessor platform must execute in order that the cumulative utilization of τ will fall within the utilization bound of this platform according to Equation 3.* ■

Suppose that we are given a certain number of processors m_o, upon which to execute periodic task system τ. At what minimum speed $s(\tau, m_o)$ should we execute these processors in order to be able to guarantee that all jobs of all tasks complete by their deadlines according to our utilization-bound of Equation 2? This question is addressed in the following lemma.

Lemma 1 *Suppose that periodic task system τ is to be implemented upon m_o identical processors. The minimum speed $s(\tau, m_o)$ at which these processors need to be executing is given by*

$$s(\tau, m_o) = \max \left[U_{\mathsf{max}}(\tau), \min \left(U_{\mathsf{max}}(\tau) + \frac{U_{\mathsf{sum}}(\tau) - U_{\mathsf{max}}(\tau)}{m_o}, \frac{2}{m_o} \times (U_{\mathsf{sum}}(\tau) - U_{\mathsf{max}}(\tau)) \right) \right] \quad (4)$$

Proof: First, observe that it is necessary for reasons of feasibility that the processors execute at a speed $\geq U_{\mathsf{max}}(\tau)$; else, the task in τ with utilization $U_{\mathsf{max}}(\tau)$ cannot possibly meet its deadline. Hence the first term in the "max" in Equation 4 above. It now remains to justify the second term in the "max".

If the utilization bound at $\mathsf{speed} = s(\tau, m_o)$ is defined by the first term in the "max" in Equation 3, then we must have

$$\begin{aligned} & U_{\mathsf{sum}}(\tau) \leq m_o \times s(\tau, m_o) - (m_o - 1) \times U_{\mathsf{max}}(\tau) \\ \equiv \quad & s(\tau, m_o) \geq \frac{U_{\mathsf{sum}}(\tau) + (m_o - 1) \times U_{\mathsf{max}}(\tau)}{m_o} \\ \equiv \quad & s(\tau, m_o) \geq U_{\mathsf{max}}(\tau) + \frac{U_{\mathsf{sum}}(\tau) - U_{\mathsf{max}}(\tau)}{m_o} \end{aligned} \quad (5)$$

If on the other hand the utilization bound at $\mathsf{speed} = s(\tau, m_o)$ is defined by the second term in the "max" in Equation 3, then we must have

$$\begin{aligned} & U_{\mathsf{sum}}(\tau) \leq \frac{m_o}{2} \times s(\tau, m_o) + U_{\mathsf{max}}(\tau) \\ \equiv \quad & s(\tau, m_o) \geq \frac{2}{m_o} \times (U_{\mathsf{sum}}(\tau) - U_{\mathsf{max}}(\tau)) \end{aligned} \quad (6)$$

If either Inequality 5 or Inequality 6 is satisfied, Condition 3 holds. Lemma 1 follows. ■

3.2 Operating with $\mathsf{speed} \leq 2U_{\mathsf{max}}(\tau)$

First, observe from Figure 1 that for $\mathsf{speed} \leq 2U_{\mathsf{max}}(\tau)$ (equivalently, $(U_{\mathsf{max}}(\tau)/\mathsf{speed}) \geq \frac{1}{2}$), the utilization bound *in*creases with decreasing processor speed (equivalently, with increasing $U_{\mathsf{max}}(\tau)/\mathsf{speed}$). That is, the fraction of the cumulative computing capacity of the platform that comprises the upper bound for the utilization $U_{\mathsf{sum}}(\tau)$ of task system τ *in*creases, simultaneously with the total power consumption *de*creasing. This argues in favor of operating as close to the point $(1.0, \frac{m}{2} + 1)$ in the utilization-bound curve (Figure 1) in order to obtain the most energy-efficient implementation.

In order to operate at the point $(1.0, \frac{m}{2} + 1)$ in the utilization-bound curve, (Figure 1), we would set the speed of each processor equal to $U_{\mathsf{max}}(\tau)$. How many processors at this speed would we need in order to be able to meet all deadlines? Recall that $\Phi(\tau)$ denotes the ratio $U_{\mathsf{sum}}(\tau)/U_{\mathsf{max}}(\tau)$. From Figure 1, an m-processor system operating at the point $(1.0, \frac{m}{2} + 1)$ has a utilization bound $(\frac{m}{2} + 1) \cdot \mathsf{speed}$; thus to accommodate a cumulative workload equal to $U_{\mathsf{sum}}(\tau)$, we would need

$$\left((\frac{m}{2} + 1) \times \mathsf{speed} \geq U_{\mathsf{sum}}(\tau) \right) \equiv m \geq (\Phi(\tau) - 1) \times 2$$

Since the number of processors is integral, we must have at least

$$m_1 \stackrel{\text{def}}{=} \lceil 2 \times (\Phi(\tau) - 1) \rceil \quad (7)$$

processors of speed $U_{\mathsf{max}}(\tau)$ each. The energy consumed by such a platform is given by

$$k \times m_1 \times U_{\mathsf{max}}(\tau)^3 \tag{8}$$

where k denotes the constant of proportionality in Equation 1.

However, such a platform may have some extra computing capacity due to the round-up error obtained in taking the $\lceil\ \rceil$ — this additional computing capacity also consumes energy. Hence, we should also consider the alternative system design obtained by taking

$$m_2 \stackrel{\text{def}}{=} \lfloor 2 \times (\Phi(\tau) - 1) \rfloor \tag{9}$$

processors, each of speed somewhat greater than $U_{\mathsf{max}}(\tau)$. The minimum speed per processor in this platform such that the cumulative computing capacity is $\geq U_{\mathsf{sum}}(\tau)$ is given by $s(\tau, m_2)$ in Equation 4 (Lemma 1), and the energy consumed by this platform is given by

$$k \times m_2 \times (s(\tau, m_2))^3 \tag{10}$$

where (as above) k denotes the constant of proportionality in Equation 1. The optimum platform is now determined by comparing the energy consumption as computed according to Equation 8 with the energy consumption computed according to Equation 10. We illustrate by a couple of examples.

Example 1 **(i)** Consider a periodic task system τ with $U_{\mathsf{sum}}(\tau) = 2.1$ and $U_{\mathsf{max}}(\tau) = 0.8$. For this system, $\Phi(\tau) = 2.1/0.8 = 2.2625$, and hence $2\times(\Phi(\tau)-1) = 3.25$; consequently, m_1 and m_2 are 4 and 3 respectively.

1. If each processor is executed at speed $U_{\mathsf{max}}(\tau) = 0.9$, then $m_1 = 4$ processors, each executing at speed 0.9, are needed; the energy consumed per unit time, according to Equation 8, is
$$\left[k \times 4 \times 0.8^3\right] \quad \equiv \quad \mathbf{k} \times 2.048$$
2. If $m_2 = 3$ processors are instead used (i.e., the configuration represented by Equation 10 is considered), then each processor runs at speed $s(\tau, 3) = 0.9583$, and the energy consumed per unit time is
$$\left(k \times 3 \times 0.8667^3\right) \quad \equiv \quad \mathbf{k} \times 1.9529$$

We thus see that the second configuration — **3** processors of speed **0.8667** each — is the more energy-efficient one.

(ii) Consider a periodic task system τ with $U_{\mathsf{sum}}(\tau) = 2.75$ and $U_{\mathsf{max}}(\tau) = 0.8$. For this system, $\Phi(\tau) = 2.75/0.8 = 3.4375$, and hence $2 \times (\Phi(\tau) - 1) = 4.875$; consequently, m_1 and m_2 are 5 and 4 respectively.

1. If each processor is executed at speed $U_{\mathsf{max}}(\tau) = 0.8$, then $m_1 = 5$ processors, each executing at speed 0.8, are needed; the energy consumed per unit time, according to Equation 8, is
$$\left[k \times 5 \times 0.8^3\right] \quad \equiv \quad \mathbf{k} \times 2.56$$
2. If $m_2 = 4$ processors are instead used (i.e., the configuration represented by Equation 10 is considered), then each processor runs at speed $s(\tau, 3) = 0.975$, and the energy consumed per unit time is
$$\left(k \times 4 \times 0.975^3\right) \quad \equiv \quad \mathbf{k} \times 3.707$$

We thus see that the first configuration — **5** processors of speed **0.8** each — is the more energy-efficient one in this case.

■

3.3 Operating with $\mathsf{speed} > 2U_{\mathsf{max}}(\tau)$

In this section, we will show that the most energy-efficient configuration does not occur for $\frac{U_{\mathsf{max}}(\tau)}{\mathsf{speed}} < \frac{1}{2}$ (from this, it will follow that the configuration identified above in Section 3.2 is the most energy-efficient one, and we will be done). To prove this, we will derive a contradiction by assuming that the most energy-efficient implementation occurs when all processors execute at a speed s such that $\frac{U_{\mathsf{max}}(\tau)}{s} < \frac{1}{2}$. In that case, it follows from Equation 2 that τ will always meet all deadlines on m processors of speed s each provided

$$\begin{aligned} & U_{\mathsf{sum}}(\tau) \leq m \cdot s - (m-1)U_{\mathsf{max}}(\tau) \\ \equiv \qquad & s \geq \frac{U_{\mathsf{sum}}(\tau) - U_{\mathsf{max}}(\tau)}{m} + U_{\mathsf{max}}(\tau) \end{aligned} \tag{11}$$

If all processors are run at this minimum speed the total power consumed when τ is implemented on m identical processors satisfies

$$\mathsf{Power}(m) \ \propto \ m \times \left(\frac{U_{\mathsf{sum}}(\tau) - U_{\mathsf{max}}(\tau)}{m} + U_{\mathsf{max}}(\tau)\right)^3. \tag{12}$$

The value of m that minimizes the power consumed can be obtained by taking the derivative of the right-hand side of Expression (12) with respect to m, and setting the resulting expression equal to zero. Solving for m, we get

$$m = 2 \times \left(\frac{U_{\mathsf{sum}}(\tau) - U_{\mathsf{max}}(\tau)}{U_{\mathsf{max}}(\tau)}\right). \tag{13}$$

Substituting this value of m [2] back in Equation 11, we conclude that s — the value of speed that minimizes the energy consumption — is equal to $\frac{3}{2} \cdot U_{\mathsf{max}}(\tau)$, from which it follows that $\frac{U_{\mathsf{max}}(\tau)}{s} = \frac{2}{3}$. However, this contradicts our assumption that the most energy-efficient implementation occurs when all processors are executing at a speed s for which $\frac{U_{\mathsf{max}}(\tau)}{s} < \frac{1}{2}$.

4 Conclusions

One of the benefits of multiprocessor implementations of embedded real-time systems is energy efficiency: since the energy consumed by a CMOS circuit is approximately proportional to the square of the

[2] We cannot always implement our system using exactly this many processors, since the m in Equation 13 may not be an integer. However, we show can show using the calculus that $\mathsf{Power}(m)$ monotonically *de*creases for m between 1 and this value, and *in*creases for m beyond this value. Hence to determine the *integral* m minimizing value of $\mathsf{Power}(m)$, it suffices to consider the two integers $\lfloor m \rfloor$ and $\lceil m \rceil$.

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 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 of independent periodic tasks;
- 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 *fixed-priority* scheduling algorithms.

For such systems, we have derived an algorithm for optimally synthesizing multiprocessor implementations of given real-time workloads such that the resulting system minimizes the amount of energy consumed.

References

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

[2] S. Baruah. Utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. Technical Report TR03-022, Department of Computer Science, The University of North Carolina, June 2003.

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

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

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

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

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

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

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

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

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

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

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

[14] T. Mudge. Power: A first class design constraint for future architectures. *IEEE Computer*, 34(4):52–58, April 2001.

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

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

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