Power Minimization for Parallel Real-Time Systems with Malleable Jobs and Homogeneous Frequencies

Antonio Paolillo
Université Libre de Bruxelles
antonio.paolillo@ulb.ac.be

Joël Goossens
Université Libre de Bruxelles
joel.goossens@ulb.ac.be

Nathan Fisher
Wayne State University
fishern@wayne.edu

Pradeep M. Hettiarachchi
Wayne State University
pradeepmh@wayne.edu

***Abstract*—In this work, we investigate the potential benefit of parallelization for both meeting real-time constraints and minimizing power consumption. We consider malleable Gang scheduling of implicit-deadline sporadic tasks upon multiprocessors. By extending schedulability criteria for malleable jobs to DVFS-enabled multiprocessor platforms, we are able to derive an *offline* polynomial-time optimal processor/frequency-selection algorithm. Simulations of our algorithm on randomly generated task systems executing on platforms having until 16 processing cores show that the theoretical power consumption is up to 36 times better than the optimal non-parallel approach.**

I. Introduction

Power-aware computing is at the forefront of embedded systems research due to market demands for increased battery life in portable devices and decreasing the carbon footprint of embedded systems in general. The drive to reduce system power consumption has led embedded system designers to increasingly utilize multicore processing architectures. An oft-repeated benefit of multicore platforms over computationally-equivalent single-core platforms is increased power efficiency and thermal dissipation [1]. For these power benefits to be fully realized, a computer system must possess the ability to parallelize its computational workload across the multiple processing cores. However, parallel computation often comes at a cost of increasing the total computation that the system must perform due to communication and synchronization overhead of the cooperating parallel processes. In this paper, we explore the trade-off between parallelization of real-time applications and savings in the power consumption.

Obtaining power efficiency for real-time systems is a non-trivial problem due to the fact that processor power-management features (e.g., clock throttling/gating, dynamic voltage/frequency scaling, etc.) often increase the execution time of jobs in order to reduce system power consumption; the increased execution time for jobs naturally puts additional temporal constraints on a real-time system. Job-level parallelism can potentially help reduce these constraints by distributing the computation to reduce the elapsed execution time of a parallel job. However, the trade-offs between parallelism, increased communication/synchronization overhead, and power reduction form a complicated and non-linear relationship.

In this paper, we address the above problem for implicit-deadline parallel sporadic tasks executing upon a multicore platform with only *global* voltage/frequency scaling capabilities. That is, all the cores on the multicore chip are constrained to execute at the same rate. For example, the Intel Xeon E3-1200 processor has such a constraint on the voltage and frequency [2]; DVFS is only possible in a package-granularity. In the past, researchers have considered the problem of determining the optimal global frequency for such systems for *non*-parallel real-time tasks (see Devadas and Aydin [3]). Parallelism contributes an additional dimension to this problem in that the system designer must also choose what is the optimal number of concurrent processors that a task should use to reduce power and meet its deadline. Our research addresses this challenge by proposing an (offline) polynomial-time algorithm for determining the optimal frequency and number of active cores for a set of parallel tasks executing upon a processing platform with homogeneous frequencies. We use a previously-proposed online scheduling algorithm by Collette et al. [4] to schedule the parallel jobs once the frequency and active core allocation has been determined.

The contributions can be summarized as follows:

- We generalize the parallel task schedulability test of Collette et al. [4] for a processing platform that may choose *offline* its operating frequency.
- We propose an offline polynomial-time algorithm for determining the optimal operating frequency and number of active cores. Given n tasks and m cores, our algorithm requires $\mathcal{O}(mn^2 \log_2^2 m)$ time.
- We illustrate the power savings of a parallel scheduling approach by comparing it against the optimal non-parallel homogeneous scheduling algorithm (via simulations of randomly-generated task systems).

The main objective of this research is to provide a theoretical evaluation of the potential reduction in system power consumption that could be obtained by exploiting parallelism of real-time applications. As we will see in the next sections, significant reductions in system power are possible even when the degree of parallelism is limited. Our current on-going work in evaluating parallel implementations of real-time applications upon an actual hardware testbed is primarily motivated by the power savings observed in the simulations of this paper.

II. Related Work

There are two main models of parallel tasks (i.e., tasks that may use several processors *simultaneously*): the *Gang* [4], [5], [6], [7] and the *Thread* model [8], [9], [10], [11]. With the Gang model, all parallel instances of a same task start and stop using the processors *synchronously* (*i.e.*, at the exact same time). On the other hand, with the Thread model, there is no such constraint. Hence, once a thread has been released, it can be executed on the processing platform independently of the execution of the other threads.

Most real-time research about energy saving has assumed sequential model of computation. For example, Baruah and Anderson [12] explicitly state that *" [...] in the job model*

typically used in real-time scheduling, 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 [...]".* In this research, we push the potential energy savings further by removing this constraint and allowing each job to be executed *synchronously* on several processing cores.

Few research has addressed both real-time parallelization and power-consumption issues. Kong et al. [13] explored the trade-offs between power and degree of parallelism for *non*-real-time jobs. Recent work by Cho et al. [14] have developed processor/speed assignment algorithms for real-time parallel tasks when the processing platform allows each processor to execute at different speed. In contrast, this work considers some restrictions on parallel processing (e.g., limited processor speedup) and power management (e.g., a single global operating frequency) that exist in many of today's multicore architectures, but are not considered in these previous papers.

III. Models

A. Parallel Job Model

In real-time systems, a job J_ℓ is characterized by its *arrival time* A_ℓ, *execution requirement* E_ℓ, and relative deadline D_ℓ. The interpretation of these parameters is that for each job J_ℓ, the system must schedule E_ℓ units of execution on the processing platform in the interval $[A_\ell, A_\ell + D_\ell)$. Traditionally, most real-time systems research has assumed that the execution of J_ℓ must occur sequentially (i.e., J_ℓ may not execute concurrently with itself on two — or more — different processors). However, in this paper, we deal with jobs which may be executed on different processors at the very same instant, in which case we say that *job parallelism* is allowed. Various kind of task models exist; Goossens et al. [6] adapted parallel terminology [15] to real-time jobs as follows.

Definition 1 (Rigid, Moldable and Malleable Job)**.** *A* job *is said to be (i)* rigid *if the number of processors assigned to this job is specified externally to the scheduler a priori, and does not change throughout its execution; (ii)* moldable *if the number of processors assigned to this job is determined by the scheduler, and does not change throughout its execution; (iii)* malleable *if the number of processors assigned to this job can be changed by the scheduler during the job's execution.*

As a starting point for investigating the trade-off between the power consumption and parallelism in real-time systems, we will work with the malleable job model in this paper.

B. Parallel Task Model

In real-time systems, jobs are generated by tasks. One general and popular real-time task model is the *sporadic task model* [16] where each sporadic task τ_i is characterized by its *worst-case execution time* e_i, *task relative deadline* d_i, and *minimum inter-arrival time* p_i (also called the task's *period*). A task τ_i generates an infinite sequence of jobs $J_1, J_2, \ldots$ such that: 1) J_1 may arrive at any time after system start time; 2) successive jobs must be separated by at least p_i time units (i.e., $A_{\ell+1} \geqslant A_\ell + p_i$); 3) each job has an execution requirement no larger than the task's worst-case execution time (i.e., $E_\ell \leqslant e_i$); and 4) each job's relative deadline is equal to the the task relative deadline (i.e., $D_\ell = d_i$). A useful metric of a task's computational requirement upon the system is *utilization* denoted by u_i and computed by e_i/p_i. In this paper, as we deal with parallel tasks that could be executed on several processors *synchronously*, systems having a utilization value greater than 1 could still be schedulable (i.e., $u_i > 1$ is permitted). Other useful specific values are $u_{\max} \stackrel{\text{def}}{=} \max_{i=1}^{n}\{u_i\}$ and $u_{\text{sum}} \stackrel{\text{def}}{=} \sum_{i=1}^{n} u_i$.

A collection of sporadic tasks $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_n\}$ is called a *sporadic task system*. In this paper, we assume a common subclass of sporadic task systems called *implicit-deadline sporadic task systems* where each $\tau_i \in \tau$ must have its relative deadline equal to its period (i.e., $d_i = p_i$). Finally, the scheduler we use restricts periods and execution times to positive integer values, *i.e.*, $e_i, p_i \in \mathbb{N}_{>0}$.

At the task level, the literature distinguishes between at least two kinds of parallelism: *Multithread* and *Gang*. In *Gang* parallelism, each task corresponds to $e \times k$ rectangle where e is the execution time requirement and k the number of required processors with the restriction that the k processors execute task in unison [7]. In this paper, we assume malleable Gang task scheduling (that is, tasks generating malleable jobs); Feitelson et al. [17] describe how a malleable job may be implemented.

Due to the overhead of communication and synchronization required in parallel processing, there are fundamental limitations on the speedup obtainable by any real-time job. Assuming that a job J_ℓ generated by task τ_i is assigned to k_ℓ processors for parallel execution over some t-length interval, the speedup factor obtainable is denoted by γ_{i,k_ℓ}. The interpretation of this parameter is that over this t-length interval J_ℓ will complete $\gamma_{i,k_\ell} \times t$ units of execution. We let $\Gamma_i = (\gamma_{i,0}, \gamma_{i,1}, \ldots, \gamma_{i,m}, \gamma_{i,m+1})$ denote the multiprocessor speedup vector for jobs of task τ_i (assuming m identical processing cores). The values $\gamma_{i,0} \stackrel{\text{def}}{=} 0$ and $\gamma_{i,m+1} \stackrel{\text{def}}{=} \infty$ are sentinel values used to simplify the algorithm of Section V. Throughout the rest of the paper, we will characterize a parallel sporadic task τ_i by (e_i, p_i, Γ_i).

We apply the following two restrictions on the multiprocessor speedup vector:

- *Sub-linear speedup ratio [13]*: $1 < \frac{\gamma_{i,j'}}{\gamma_{i,j}} < \frac{j'}{j}$ where $0 < j < j' \leqslant m$.
- *Work-limited parallelism [4]*: $\gamma_{i,(j'+1)} - \gamma_{i,j'} \leqslant \gamma_{i,(j+1)} - \gamma_{i,j}$ where $0 \leqslant j < j' < m$.

The sub-linear speedup ratio restriction represents the fact that no task can truly achieve an ideal or better than ideal speedup due to the overhead in parallelization. It also requires that the speedup factor strictly increases with the number of processors. The work-limited parallelism restriction ensures that the overhead only increases as more processors are used by the job. These restrictions place realistic bounds on the types of speedups observable by parallel applications. There are no other restriction on the actual values of these speedup parameters, *i.e.*, $\forall 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m : \gamma_{i,j} \in \mathbb{R}_{>0}$. An example of two speedup vectors, used in our simulations, is given by Figure 3.

C. Power/Processor Model

The parallel sporadic task system τ executes upon a multiprocessor platform with $m \in \mathbb{N}_{>0}$ identical-speed processing cores. The processing platform is enabled with both dynamic power management (DPM) and dynamic voltage and frequency scaling (DVFS) capabilities. With respect to DPM capabilities, we assume that the processing platform has the ability to turn off any number of cores between 0 and $m-1$. For DVFS capabilities, in this work, we assume that there is a system-wide homogeneous frequency $f > 0$ (where f is drawn from the positive continuous range – i.e., $f \in \mathbb{R}_{>0}$) which indicates the frequency at which all cores are executing at any given moment. The power function $P(f,k)$ indicates the power dissipation rate of the processing platform when executing with k active cores at a frequency of f. We assume that $P(f,k)$ is a non-decreasing, convex function.

The interpretation of the frequency is that if τ_i is executing job J_ℓ on k_ℓ processors at frequency f over a t-length interval then it will have executed $t \times \gamma_{i,k_\ell} \times f$ units of computation. The total energy consumed by executing k cores over the t-length interval at frequency f is $t \times P(f,k)$.

Since we are considering a single system-wide homogeneous frequency, a natural question is: *does the ability to dynamically change frequencies during execution contribute towards our goal of reducing power and/or meeting job deadlines*? We can show that the answer to the question is "no"; it turns out that there exists a single optimum frequency for a given set of malleable real-time tasks:

Property 1. *[Obtained by extension of Aydin [18], and Ishihara and Yasuura [19]] In a multiprocessor system with global homogeneous frequency in a continuous range, choosing dynamically the frequency is not necessary for optimality in terms of minimizing total consumed energy.*

D. Scheduling Algorithm

In this paper, we use a scheduling algorithm originally developed for non-power-aware parallel real-time systems called the *canonical parallel schedule* [4]. The canonical scheduling approach is optimal for implicit-deadline sporadic real-time tasks with work-limited parallelism and sub-linear speedup ratio upon an identical multiprocessor platform (i.e., each processor has identical processing capabilities and speed). In this paper, we consider also an identical multiprocessor platform, but permit both the number of active processors and homogeneous frequency f for all active processors to be chosen prior to system run-time. In this subsection, we briefly define the canonical scheduling approach with respect to our power-aware setting.

Assuming the processor frequencies are identical and set to a fixed value f, it can be noticed that a task τ_i requires more than k processors simultaneously if $u_i > \gamma_{i,k}\, f$; we denote by $k_i(f)$ the largest such k (meaning that $k_i(f)$ is the smallest number of processor(s) such that the task τ_i is schedulable on $k_i(f)+1$ processors at frequency f):

$$k_i(f) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } u_i \leq \gamma_{i,1}\, f \\ \max_{k=1}^{m}\{k \mid \gamma_{i,k}\, f < u_i\}, & \text{otherwise.} \end{cases} \quad (1)$$

The *canonical schedule* fully assigns $k_i(f)$ processor(s) to τ_i and at most one additional processor is partially assigned (see [4] for details). This definition extends the original definition of k_i from non-power-aware parallel systems [4].

As an example, let us consider the task system $\tau = \{\tau_1, \tau_2\}$ to be scheduled on $m = 3$ processors with $f = 1$. We have $\tau_1 = (6, 4, \Gamma_1)$ with $\Gamma_1 = (1.0, 1.5, 2.0)$ and $\tau_2 = (3, 4, \Gamma_2)$ with $\Gamma_2 = (1.0, 1.2, 1.3)$. Notice that the system is infeasible at this frequency if job parallelism is not allowed since τ_1 will never meet its deadline unless it is scheduled on at least two processors (i.e., $k_1(1) = 1$). There is a feasible schedule if the task τ_1 is scheduled on two processors and τ_2 on a third one (i.e., $k_2(1) = 0$).

E. Problem Definition

We are now prepared to formally state the problem addressed in this paper.

> Given a malleable, implicit-deadline sporadic task system τ, DVFS/DPM-enabled processor with m cores, and canonical parallel scheduling, we will determine (offline) the optimal choice of system-wide homogeneous frequency f and number of active cores k such that $P(f,k)$ is minimized and no task misses a deadline in the canonical schedule.

To solve this problem, we will introduce an algorithm, based on the schedulability criteria of the *canonical schedule*, to determine the optimal *offline* frequency/number of processors combination and evaluate our solution over simulations.

IV. Preliminary Results

In this section, we restate the schedulability criteria for canonical scheduling under homogeneous frequencies and show that the criteria is *sustainable* (i.e., a schedulable system remains schedulable even if the frequency or number of active cores is increased). We will use these results in the next section to develop an algorithm for determining the optimal choice of number of active cores (k) and system-wide frequency (f).

A. Schedulability Criteria of Malleable Task System with Homogeneous Frequency

As we gave in Section III-C the mathematical interpretation of the parameter f over system execution, it is easy to adapt the schedulability criteria of [4] to a power-aware schedule. Indeed, we just have to replace u_i by $\frac{u_i}{f}$ in schedulability conditions. This lead us to the following theorem.

Theorem 1 (extended from Collette et al. [4])**.** *A necessary and sufficient condition for an implicit-deadlines sporadic malleable task system τ respecting* sub-linear speedup ratio *and* work-limited parallelism*, to be schedulable on m processors at frequency f is given by:*

$$\begin{cases} \max_{i=1}^{n}\{k_i(f)\} < m \\ \\ \sum_{i=1}^{n}\left(k_i(f) + \frac{u_i - \gamma_{i,k_i(f)} f}{(\gamma_{i,k_i(f)+1} - \gamma_{i,k_i(f)}) f}\right) \leqslant m\ . \end{cases} \quad (2)$$

Given the above schedulability criteria, we can easily obtain an algorithm for determining in $O(n \log m)$ – with $k_i(f)$ computed by binary search over m values – whether the set of n tasks on m identical processing cores running all at frequency f is schedulable.

B. Sustainability of the Frequency for the Schedulability

In this section, we will prove an important property of our framework: sustainability. We will prove that if a system is schedulable, then increasing the homogeneous frequency will maintain the schedulability of the system. This implies that there is a unique minimum frequency for a couple task system/number of processors to be schedulable. The algorithm introduced in Section V will use this property to efficiently search for this optimal minimum frequency. In order to prove that property, we will need several new notations and concepts.

Definition 2 (Minimum Number of Processors for a Task)**.** *For any $\tau_i \in \tau$, the minimum number of processors is denoted by:*

$$M_i(f) \stackrel{\text{def}}{=} k_i(f) + \frac{u_i - \gamma_{i,k_i(f)} f}{(\gamma_{i,k_i(f)+1} - \gamma_{i,k_i(f)}) f}$$

Therefore, we can define the same notion system-wide:

$$M_\tau(f) \stackrel{\text{def}}{=} \sum_{i=1}^{n} M_i(f)$$

Figure 1 illustrates the behaviour of $k_i(f)$ and $M_i(f)$. (The dotted curved line corresponds to $M_i(f)$ and the solid-step line corresponds to $k_i(f)$). Based on this definition, the schedulability criteria (2) becomes:

$$\max_{i=1}^{n}\{k_i(f)\} < m \quad \wedge \quad M_\tau(f) \leqslant m \ . \tag{3}$$

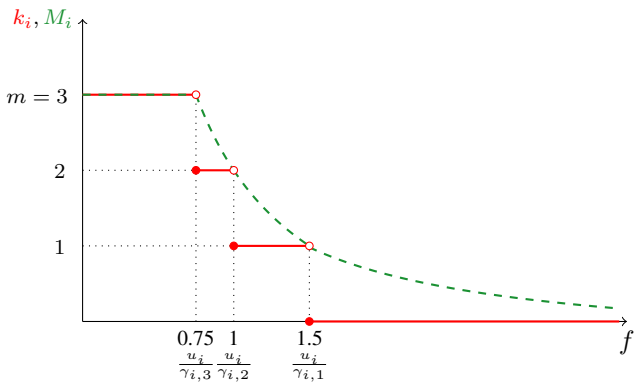


Fig. 1: Plot of k_i and M_i for $m = 3, \tau_i = (6, 4, \langle 1.0, 1.5, 2.0\rangle)$

This definition will be useful in the following main theorem.

Theorem 2. *The schedulability of the system is sustainable regarding the frequency,* i.e.*, increasing the frequency preserves the system schedulability.*

Proof Sketch: Due to space limitations, we only provide a sketch of the theorem proof. Observe that both $k_i(f)$ and $M_i(f)$ (and also $M_\tau(f)$) are monotonically non-increasing in f. Thus, if the conditions of Equation 3 are satisfied for a given f, they will continue to be satisfied for any $f' \geqslant f$ since $M_i(f') \leqslant M_i(f)$ and $k_i(f') \leqslant k_i(f)$. □

V. Optimal Processor/Frequency-Selection Algorithm

A. Algorithm Description

Theorem 2 implies that there is a minimum frequency for the system to be schedulable. The challenge of this section is to inverse the function $M_\tau(f)$ in order to have an expression of the frequency depending of the number of processors m. This is not trivial because $M_\tau(f)$ is the sum of a continuous term and discontinuous term (the expression of $k_i(f)$). Furthermore, since f is potentially any real positive number, exhaustive search is not an applicable approach and the optimal value of f must be determined analytically. We present an algorithm that computes the exact optimal minimum frequency for a particular task system τ and a number of processors m in $O(n^2 \log_2^2 m)$ time (see Algorithm 1). We use this algorithm in conjunction of the power function $P(f, k)$ to determine the optimal number of active cores and system-wide frequency.

Consider fixing each $k_i(f)$ term (for $i = 1, \ldots, n$) with values $\bar{\kappa}_1, \bar{\kappa}_2, \ldots, \bar{\kappa}_n \in \{0, 1, \ldots, m-1\}$, each corresponding to the number of processors potentially assigned to each task τ_i at a frequency f. Then, from Definition 3 we can replace $k_i(f)$ by $\bar{\kappa}_i$ in schedulability inequations (3). The first condition is always true because by choice of $\bar{\kappa}_i$. For the second condition:

$$\sum_{i=1}^{n} \bar{\kappa}_i + \frac{u_i - \gamma_{i,\bar{\kappa}_i} f}{(\gamma_{i,\bar{\kappa}_i+1} - \gamma_{i,\bar{\kappa}_i})\, f} \leqslant m \ .$$

By isolating f, this is equivalent to:

$$f \geqslant \frac{\sum_{i=1}^{n} \frac{u_i}{\gamma_{i,\bar{\kappa}_i+1} - \gamma_{i,\bar{\kappa}_i}}}{m - \sum_{i=1}^{n} \left(\bar{\kappa}_i - \frac{\gamma_{i,\bar{\kappa}_i}}{\gamma_{i,\bar{\kappa}_i+1} - \gamma_{i,\bar{\kappa}_i}}\right)} \stackrel{\text{def}}{=} \Psi_\tau(m, \bar{\kappa}) \ ,$$

where $\bar{\kappa} \stackrel{\text{def}}{=} \langle \bar{\kappa}_1, \bar{\kappa}_2, \ldots, \bar{\kappa}_n \rangle$. We have derived a lower bound on the frequency that satisfies Equation 3 given fixed $\bar{\kappa}$.

Notice in solving for f in the above paragraph, it is possible that we have chosen values for $\bar{\kappa}$ that do not correspond to the $k_i(f)$ values. If so, then the value returned by $\Psi_\tau(m, \bar{\kappa})$ may not correspond to a frequency for which τ is schedulable. To address this problem, we may symmetrically also fix a frequency f and determine the corresponding values of $k_i(f)$ according to Equation 1. Let $\bar{\kappa}_\tau(f)$ be the vector $\langle k_1(f), k_2(f), \ldots, k_n(f)\rangle$. For all $\tau_i \in \tau$, $k_i(f) < m$ if $f > u_i/\gamma_{i,m}$. Thus, if $f > \max_{i=1}^{n}\{u_i/\gamma_{i,m}\}$ and the following inequality is satisfied, then Theorem 1 and τ is schedulable given frequency f.

$$f \geqslant \Psi_\tau(m, \bar{\kappa}(f)). \tag{4}$$

Recall that our goal is to minimize the non-decreasing function $P(f, k)$. Therefore, we want the smallest $f > \max_{i=1}^{n}\{u_i/\gamma_{i,m}\}$ that satisfies Inequation 4 which leads to the following definition.

Definition 3 (Minimum optimal frequency)**.** *The minimum optimal frequency of a system τ schedulable on m processors is denoted as $f_{\min}^{(\tau,m)}$. Formally,*

$$f_{\min}^{(\tau,m)} \stackrel{\text{def}}{=} \min\{f \in \mathbb{R}_{>0} \mid \mathsf{schedulable}(\tau, m, f)\} \ ,$$

where $\mathsf{schedulable}(\tau, m, f)$ *is* true *if and only if (3) holds. Note this minimum corresponds to when (4) achieves equality.*

Consider now taking the inverse of function $k_i(f)$:

$$k_i^{-1}(\kappa) \stackrel{\text{def}}{=} \begin{cases} \{f \mid \frac{u_i}{\gamma_{i,\kappa+1}} \leqslant f < \frac{u_i}{\gamma_{i,\kappa}}\} & \text{if } 0 < \kappa < m \\ [\frac{u_i}{\gamma_{i,1}}, \infty) & \text{otherwise.} \end{cases} \tag{5}$$

We can see that k_i^{-1} is indeed the inverse function of k_i by looking at the behaviour of k_i illustrated by Figure 1. Furthermore, by the definition of k_i^{-1} we can see that $k_i(f)$ remains fixed for all f over the interval $[\frac{u_i}{\gamma_{i,\kappa+1}}, \frac{u_i}{\gamma_{i,\kappa}})$ for each different $\kappa : 0 \leqslant \kappa \leqslant m$. Using this observation and the fact that any

schedulable frequency f is greater than $\max_{i=1}^{n}\{u_i/\gamma_{i,m}\}$, we only need to check Inequality (4) at values of f from the set $\bigcup_{\kappa=1}^{m}\{\frac{u_i}{\gamma_{i,\kappa}}\}$ to determine the maximum κ_i such that τ is schedulable. The main idea of Algorithm 1 is that we determine for each $\tau_i \in \tau$ the maximum value of κ_i for the system to be schedulable. Using these determined values of κ_i's, we can evaluate $\Psi_\tau(m, \bar{\kappa} \stackrel{\text{def}}{=} \langle \bar{\kappa}_1, \ldots, \bar{\kappa}_n \rangle)$ to obtain the solution $f_{\min}^{(\tau,m)}$.

Algorithm 1: minimumOptimalFrequency(τ, m)

for $i \in \{1, 2, \ldots, n\}$ **do**
  if schedulable$(\tau, m, \frac{u_i}{\gamma_{i,m}})$ **then**
    $\bar{\kappa}_i \leftarrow m - 1$
  else
    $\bar{\kappa}_i \leftarrow \min_{\kappa=0}^{m-1}\{\kappa \mid \textbf{not}\ \text{schedulable}(\tau, m, \frac{u_i}{\gamma_{i,\kappa+1}})\}$
$\bar{\kappa} \stackrel{\text{def}}{=} \langle \bar{\kappa}_1, \bar{\kappa}_2, \ldots, \bar{\kappa}_n \rangle$
return $\Psi_\tau(m, \bar{\kappa})$

To compute schedulable(τ, m, f), we determine the value of $k_i(f)$ from frequency f according to (1), which can be obtained in $O(\log_2 m)$ time by binary search over m values. To calculate $k_i(f)$ for all $\tau_i \in \tau$ and sum every $M_i(f)$ terms, the total time complexity of the schedulability test is $O(n \log_2 m)$.

In Algorithm 1 aimed at calculating $f_{\min}^{(\tau,m)}$, the value of $\bar{\kappa}_i$ can also be found by binary search and takes $O(\log_2 m)$ time to compute. This is made possible by the sustainability of the system regarding the frequency (proved by Theorem 2). Indeed, if τ is schedulable on m processors with $f = \frac{u_i}{\gamma_{i,\kappa_i+1}}$, then it's also schedulable with $f = \frac{u_i}{\gamma_{i,\kappa_i}} > \frac{u_i}{\gamma_{i,\kappa_i+1}}$.

In order to calculate the complete vector $\bar{\kappa}$, there will be $O(n \log_2 m)$ calls to the schedulability test. Since computing Ψ_τ is linear-time when the vector $\bar{\kappa}$ is already stored in memory, the total time complexity to determine the optimal schedulable frequency for a given number of processors is $O(n^2 \log_2^2 m)$. In order to determine the optimal combination of frequency and number of processors, we simply iterate over all possible number of active processors $\ell = 1, 2, \ldots, m$ executing Algorithm 1 with inputs τ and ℓ. We return the combination that results in the minimum overall power-dissipation rate (computed with $P(f_{\min}^{(\tau,\ell)}, \ell)$). Thus, the overall complexity to find the optimal combination is $O(mn^2 \log_2^2 m)$.

B. Proof of Correctness

Theorem 3. *Algorithm 1 returns* $f_{\min}^{(\tau,m)}$.

Proof Sketch: Due to space considerations, we only give a brief sketch of the proof. By the arguments in Section V-A, Algorithm 1 chooses for each $\tau_i \in \tau$ the maximum $\bar{\kappa}_i$ such that the system is schedulable. Let f' equal $\Psi_\tau(m, \bar{\kappa})$ where $\bar{\kappa}$ is the values of $\bar{\kappa}_i$ determined in Algorithm 1. Note that for all $\tau_i \in \tau$, $M_i(f')$ equals $\bar{\kappa}_i + \frac{u_i - \gamma_{i,\bar{\kappa}_i} f'}{(\gamma_{i,\bar{\kappa}_i+1} - \gamma_{i,\bar{\kappa}_i}) f'}$. Therefore, it must be that

$$f' = \Psi_\tau(m, \bar{\kappa}(f')).$$

Since (4) has reached equality with f', this is the smallest frequency such that τ is schedulable. Therefore, f' must be equal to $f_{\min}^{(\tau,m)}$. □

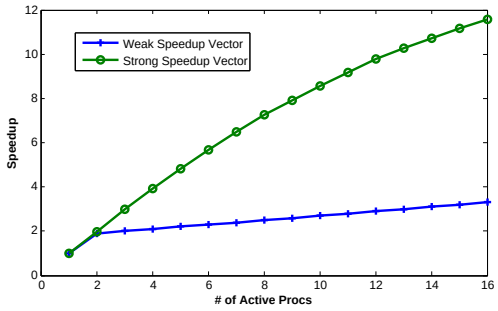


Fig. 3: Speedup Vectors for *strong* and *weak* parallelized systems.

VI. Simulations

In order to investigate the potential utility of parallelism upon power consumption, we have evaluated our algorithm with random simulations. In this section, we describe and discuss the high-level overview of the methodology employed in our evaluation and the results obtained from our simulations.

A. Methodology Overview

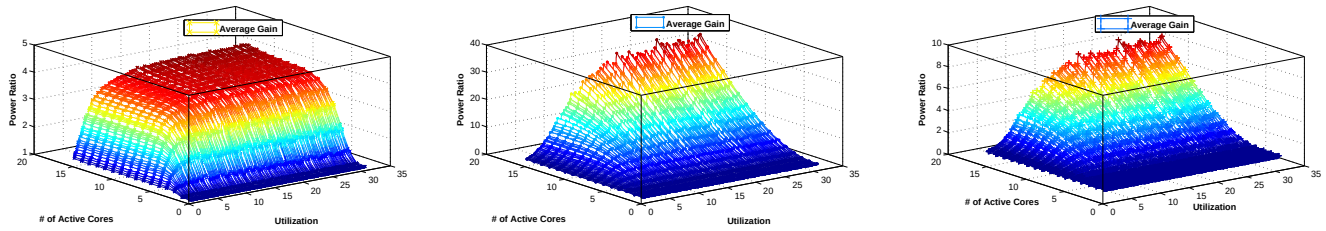
The details for each step of our simulation framework are the following:

1) Random Task Sets Generation: We randomly generate execution times e_i and periods p_i with the Stafford's `RandomVectorsFixedSum` algorithm [20] (modified to permit $u_i > 1$) that is proven [21] to generate uniformly-distributed multiprocessor task systems (i.e., $u_{\text{sum}} > 1$).

2) Speedup Vectors Values: To fix the speedup vectors of these task systems, we have modeled the execution behavior of two kinds of parallel systems: one not affected significantly by communication and I/O overheads, called the *strong parallelized system* (SPS), and another one heavily affected by communication and I/O overheads, called the *weak parallelized system* (WPS). Values of these two vectors are shown on Figure 3. Notice that both vectors respects *sub-linear speedup ratio* and *work-limited parallelism*, as defined in Section III.

3) Minimum Frequency Determination: For each system generated in Step 1, we compute three distinct frequency values: the first two values are the minimum frequencies for the system to be schedulable when tasks are *strongly* parallelized and *weakly* parallelized. To compute these two values, we use Algorithm 1. The third value, computed to show the difference with non-parallelized task systems, is the minimum frequency required for a non-parallel scheduling algorithm (referred to as SEQ) to schedule the same task system denoted as $f_{\text{seq}} \stackrel{\text{def}}{=} \max\left(u_{\max}, \frac{u_{\text{sum}}}{m}\right)$. Recall that u_i may exceed 1.

4) Power Dissipation Rate: We define the total power consumption, introduced in Section III-C, by dynamic and static power portions to closely resembles a physical processing platforms [22]. The static (leakage) power consumption of a processor can be as high as 42% of total power and depends on many factors [23]. In this research, we let the processor static power equal 15% of the total dynamic power when running at unit frequency. For this simulation, we use, $P(f, k) = f^3 k + 0.15 \times k$, where f is the processing frequency and k is the number of active cores; the two additive



(a) Power savings for WPS over SEQ. (b) Power savings for SPS over SEQ. (c) Power savings for SPS over WPS.

Fig. 2: Average power savings of parallelized systems.

terms represent dynamic and static power, respectively, and both depend on k. Notice that it respects constraint of the function as defined in Section III-C. Power consumption is then computed in function for each of the frequency/number of actives cores choices taken in Step 3.

5) Results Comparison: Using the random-task generator, we generate task systems with 8 tasks. The total system utilization is varied from 1.5 to 32.0 by 0.1 increments and number of available cores are varied from 1 to 16. The simulation runs for each task system/maximum number of cores pair. For each utilization point, we store the exact frequency returned by our algorithm and the associated power consumed.

Our frequency/processor-selection algorithm is compared against the power required by an optimal non-parallel real-time scheduling approach. The power *gains* of parallelisation are plotted in Figure 2 and computed by taking the quotient between power consumed by the system in sequential mode and in parallel (malleable) mode, each time by taking the minimum frequency computed in Step 3. This allows us to manipulate *relative* gain of the parallel paradigm over the sequential one. Each data point is the average power saving for 100 different randomly-generated task systems.

B. Results & Discussion

Figures 2a, 2b, and 2c display the power savings obtained from our simulations. Each point represents the average of power saving ratios over 100 randomly generated systems each having a fixed total utilization u_{sum} and maximum m number of processing cores. The *z-values* of the plots are then: $\mathbb{E}[\frac{P(f_{\text{seq}},k_{\text{seq}})}{P(f_{\text{wps}},k_{\text{wps}})}]$ (Fig. 2a), $\mathbb{E}[\frac{P(f_{\text{seq}},k_{\text{seq}})}{P(f_{\text{sps}},k_{\text{sps}})}]$ (Fig. 2b) and $\mathbb{E}[\frac{P(f_{\text{wps}},k_{\text{wps}})}{P(f_{\text{sps}},k_{\text{sps}})}]$ (Fig. 2c) where, in the setting x, f_x represents the minimum optimal frequency and k_x represents the number of activated cores ($k_x \leqslant m$).

We can see on Fig. 2b that, even when the application is weakly parallelized (strong communications and synchronisation overheads), the power gain can be up to $4\times$ w.r.t. the sequential execution. Moreover, in the strongly parallelized setup, power gain can be up to $36\times$ w.r.t. the sequential execution (see 2a).

VII. Conclusions

In this paper, we show the benefits of parallelization for both meeting real-time constraints and minimizing power consumption. Our research suggests the potential in reducing the overall power consumption of real-time systems by exploiting job-level parallelism. In the future, we will extend our research to investigate power saving potential when the cores may execute at different frequencies and also incorporate thermal constraints into the problem. We will also consider practical implementations of parallel power-aware online schedulers into a RTOS deployed upon an actual hardware testbed.

References

[1] "The benefits of multiple cpu cores in mobile devices (white paper)," NVIDIA Corporation, Tech. Rep., 2010.

[2] "Intel xeon processor e3-1200 family datasheet," http://www.intel.com/content/www/us/en/processors/xeon/xeon-e3-1200-family-vol-1-datasheet.html.

[3] V. Devadas and H. Aydin, "Coordinated power management of periodic real-time tasks on chip multiprocessors," in *Proceedings of the First IEEE International Green Computing Conference (IGCC'10)*, August 2010.

[4] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *Information Processing Letters*, vol. 106, no. 5, pp. 180–187, 2008.

[5] V. Berten, P. Courbin, and J. Goossens, "Gang fixed priority scheduling of periodic moldable real-time tasks," in *JRWRTC 2011*, pp. 9–12.

[6] J. Goossens and V. Berten, "Gang FTP scheduling of periodic and parallel rigid real-time tasks," in *RTNS 2010*, pp. 189–196.

[7] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in *RTSS 2009*, pp. 459–468.

[8] P. Courbin, I. Lupu, and J. Goossens, "Scheduling of hard real-time multi-phase multi-thread periodic tasks," *Real-Time Systems: The International Journal of Time-Critical Computing*, vol. 49, no. 2, pp. 239–266, 2013.

[9] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *RTSS 2010*, pp. 259–268.

[10] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *RTSS 2011*, pp. 217–226.

[11] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Techniques optimizing the number of processors to schedule multi-threaded tasks," in *ECRTS 2012*, pp. 321–330.

[12] J. Anderson and S. Baruah, "Energy-efficient synthesis of EDF-scheduled multiprocessor real-time systems," *International Journal of Embedded Systems 4 (1)*, 2008.

[13] F. Kong, N. Guan, Q. Deng, and W. Yi, "Energy-efficient scheduling for parallel real-time tasks based on level-packing," in *SAC 2011*, pp. 635–640.

[14] S. Cho and R. Melhem, "Corollaries to Amdahl's law for energy," *Computer Architecture Letters*, vol. 7, no. 1, pp. 25–28, 2007.

[15] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999, ch. Scheduling Parallel Jobs on Clusters, pp. 519–533.

[16] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983.

[17] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Proceedings of the Job Scheduling Strategies for Parallel Processing*, ser. IPPS '97, 1997, pp. 1–34.

[18] H. Aydin, "Enhancing performance and fault tolerance in reward-based scheduling," Ph.D. dissertation, University of Pittsburgh, PA, 2001.

[19] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *ISLPED 1998*, pp. 197–202.

[20] R. Stafford, "Random vectors with fixed sum," 2006. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/9700

[21] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Jul., pp. 6–11.

[22] H.-S. Yun and J. Kim, "On energy-optimal voltage scheduling for fixed-priority hard real-time systems," *ACM Trans. Embed. Comput. Syst.*, 2003.

[23] J. Kao, S. Narendra, and A. Chandrakasan, "Subthreshold leakage modeling and reduction techniques," in *IEEE/ACM International Conference on Computer Aided Design*, 2002, pp. 141–148.