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

Antonio Paolillo, Joël Goossens

PARTS Research Centre
Université Libre de Bruxelles and Mangogem S.A.
{antonio.paolillo, joel.goossens}@ulb.ac.be

Pradeep M. Hettiarachchi, Nathan Fisher

Department of Computer Science
Wayne State University
{pradeepmh, fishern}@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 DPM/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 up to 16 processing cores show that the theoretical power consumption is reduced by a factor of 36 compared to 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 and/or introduce switching time overheads 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. Thus, for power-aware multicore real-time systems, an important and challenging open question is: *what is the optimal combination of job-level parallelism and processor power-management settings to minimize system power consumption while simultaneously ensuring real-time deadlines are met?*

In this paper, we address the above problem for implicit-deadline parallel sporadic tasks executing upon a multicore platform with unique *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]; dynamic voltage and frequency scaling (DVFS) is only possible in a package-granularity (*i.e.*, we have to choose one working frequency for all active cores of the processing platform). However, we also permit dynamic power management (DPM): this means that some cores of the platform can be switched-off. In summary, we allow the selection of a subset of the cores to be active and all these chosen cores must run at the same frequency.

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 exact 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 *in unison* (*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, by reducing the minimum required speed of the platform below the sequential limit, we push the potential power/energy savings further by removing this constraint and allowing each job to be executed *in unison* 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 time 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. It means that for each time units in the interval $[A_\ell, A_\ell + D_\ell)$, several units of execution of J_ℓ can be executed (corresponding to the number of processors assigned to J_ℓ). Various kind of parallel 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 requirement* 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 of the same task 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 requirement (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 *in unison* on several processors with variable speed (DVFS/DPM enabled), tasks having a utilization value greater than 1 could still be schedulable (i.e., $u_i > 1$ is permitted). Indeed, to meet its deadline a job of a task having a utilization greater than 1 must either 1) be executed in unison on several processing cores of the platform or 2) be executed at an appropriate processor speed (higher frequency). This frequency/number of cores selection trade-off is specifically the problem we explore in this paper. 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 requirements 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 $\Gamma_i \stackrel{\text{def}}{=} (\gamma_{i,0}, \gamma_{i,1}, \ldots, \gamma_{i,m}, \gamma_{i,m+1})$ 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]*:
$$1 < \frac{\gamma_{i,j'}}{\gamma_{i,j}} < \frac{j'}{j}$$
$$\text{where} \quad 0 < j < j' \leqslant m\,.$$
- *Work-limited parallelism [4]*:
$$\gamma_{i,(j'+1)} - \gamma_{i,j'} \leqslant \gamma_{i,(j+1)} - \gamma_{i,j}$$
$$\text{where} \quad 0 \leqslant j < j' < m\,.$$

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. Notice that these constraints imply that $\forall 1 \leqslant i \leqslant n, 0 \leqslant j < m : \gamma_{i,j} < \gamma_{i,j+1}$. It could be argued that this constraint is not entirely realistic: at a reasonable high number of processing cores, allocating an additional core to the task will not increase the speedup anymore. However, while our mathematical model requires that the speedup must increase with each additional processing core, the situation where adding a core does not benefit the application speedup can be modeled with $\gamma_{i,j+1} + \epsilon = \gamma_{i,j}$ where ϵ can be some arbitrarily small positive real number. Thus, this strict inequality constraint does not place any true restriction upon approximating such realistic parallel behavior.

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 2 (page 8).

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. In short, at any moment in the execution of the system, if $k \leqslant m$ processing cores are switched on and $m - k$ cores are switched off (DPM) and the homogeneous frequency of the system is set to the value f (DVFS), it means that k cores run at frequency f and $m - k$ cores *"run"* at frequency 0.

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. See section VII-A3 for an instance of this function in our simulations.

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.*

Proof. As [19] presented similar result, here we prove the property for our framework.

Although we have a proof of this property for any convex form of $P(f, k)$, for space limitation in the following, we will consider that $P(f, k) \propto f^3$ (notice that k is a constant in our analysis). Assume we have a schedule at the constant frequency f on the (multiprocessor) platform that is feasible for τ. We will show that any dynamic frequency schedule — that is also feasible for τ— consumes not less energy.

First notice that from any dynamic frequency schedule we can obtain a constant frequency schedule (which schedules the same amount of work) by applying, sequentially, the following transformation: given a dynamic frequency schedule in the interval $[a, b]$ which works at frequency f_1 in $[a, \ell)$ and at frequency f_2 in $[\ell, b]$ we can define the constant voltage such that at that frequency the executed amount of each task $\tau_i \in \tau$ remains the same as the execution in the dynamic-frequency schedule over $[a, b]$.

Without loss of generality we will consider schedule in the interval $[0, 1]$ working at the constant frequency f and the dynamic schedule working at frequency $f + \Delta$ in $[0, \ell)$ and at the frequency $f - \Delta'$ in $[\ell, 1]$. Since the transformation must preserve the amount of work completed we must have:

$$f = \ell(f + \Delta) + (1 - \ell)(f - \Delta')$$
$$\Leftrightarrow \Delta' \stackrel{\text{def}}{=} \frac{\ell\Delta}{1 - \ell} \tag{1}$$

since the extra work in $[0, \ell)$ (i.e., $\Delta\ell$) must be equal to the spare work in $[\ell, 1]$ (i.e., $\Delta'(1 - \ell)$).

Now we will compare the *relative* energy consumed by both

the schedules, i.e., we will show that

$$\ell(f+\Delta)^3 + (1-\ell)(f - \frac{\ell\Delta}{1-\ell})^3 \geqslant f^3 \qquad (2)$$

We know that $\ell(f+\Delta)^3 = \ell(f^3+3f^2\Delta+3f\Delta^2+\Delta^3)$ and $(1-\ell)(f-\frac{\ell\Delta}{1-\ell})^3 = (1-\ell)(f^3-3f^2\frac{\ell\Delta}{1-\ell}+3f\frac{\ell^2\Delta^2}{(1-\ell)^2}-\frac{\ell^3\Delta^3}{(1-\ell)^3})$.

(2) is equivalent to (by subtracting f^3 on the both sides)

$$\Delta\left[3\ell f\Delta + \ell\Delta^2 + 3f\frac{\ell^2\Delta}{(1-\ell)} - \frac{\ell^3\Delta^2}{(1-\ell)^2}\right] \geqslant 0$$

Or equivalently (dividing by $\ell\Delta$):

$$3\Delta f + \Delta^2 + 3f\frac{\ell\Delta}{(1-\ell)} - \frac{\ell^2\Delta^2}{(1-\ell)^2} \geqslant 0$$

$\Leftarrow$ ($f - \Delta' > 0$ and, by (1))

$$3\Delta f + \Delta^2 + 3\frac{\ell\Delta}{1-\ell}\frac{\ell\Delta}{(1-\ell)} - \frac{\ell^2\Delta^2}{(1-\ell)^2} \geqslant 0$$

$\Leftarrow$

$$3\Delta f + \Delta^2 + 2\frac{\ell^2\Delta^2}{(1-\ell)^2} \geqslant 0$$

which always holds because $\Delta > 0$ and $f > 0$.

To complete the proof we must show that our transformation preserves the amount of execution for each job of τ over its release and deadline. Consider the execution of a job of τ_i over some interval $[a, b]$ (where $a, b \in \mathbb{N}$) between its release and deadline where τ_i executed e_1 units over $[a, \ell)$ and e_2 over $[\ell, b]$ (where $\ell \in \mathbb{N}$). Furthermore, consider that one interval executes proportionally more of τ_i than the other interval. Without loss of generality, let $\frac{e_1}{\ell-a} > \frac{e_2}{b-\ell}$. We can show that the schedule consumes less of the processing time (and thus remains feasible in the single frequency schedule). The next subsection introduces an optimal parallel scheduler that can execute each task over every unit-length interval at a pre-specified rate. Using this scheduler, over $[a, a+1), [a+1, a+2), \ldots [\ell-1, \ell)$, we can execute $\frac{e_1}{\ell-a}$ units in each interval. Similarly, we can execute $\frac{e_2}{b-\ell}$ units in each unit-length interval of $[\ell, b]$. Since the rate of execution is higher for the first interval and lower for the second, the level of parallelism for τ_i must be greater in the first. If we instead execute $\frac{e_1+e_2}{b-a}$ over all intervals, the total amount of execution is preserved. However, the total processing required is decreased as the speed-up is concave function over the level of parallelism (due the properties of sub-linear speedup ratio and work-limited parallelism). Thus, over each job-release and deadline of τ_i, we can use a constant rate of execution. Since we have an implicit-deadline task system, this rate is maintained over all intervals.

□

As a consequence, without loss of generality, we will consider systems where the number of active cores and the homogeneous frequency is decided prior the execution of the system, *i.e.*, *offline*.

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):

$$k_i(f) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } u_i \leq \gamma_{i,1}\, f \\ \max_{k=1}^{m}\{k \mid \gamma_{i,k}\, f < u_i\}, & \text{otherwise.} \end{cases} \qquad (3)$$

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 by the canonical schedule on m processors at frequency f is given by:*

$$\begin{cases} \max_{i=1}^{n}\{k_i(f)\} < m \quad \textit{and} \\ \sum_{i=1}^{n}\left(k_i(f) + \frac{u_i - \gamma_{i,k_i(f)} f}{(\gamma_{i,k_i(f)+1} - \gamma_{i,k_i(f)})f}\right) \leqslant m \ . \end{cases} \tag{4}$$

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 (see Algorithm 1).

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 (4) becomes:

$$\max_{i=1}^{n}\{k_i(f)\} < m \qquad \text{and} \qquad M^{\tau}(f) \leqslant m \ . \tag{5}$$

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: 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 (5) 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 $k_i(f') \leqslant k_i(f)$. □

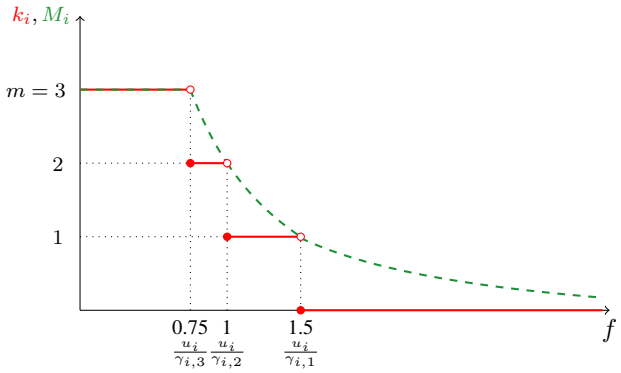


Fig. 1: Plot of k_i and M_i for $m = 3, \tau_i = (6, 4, \langle 1.0, 1.5, 2.0\rangle)$

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 we assumed that f is potentially any real positive number, the mathematical sound way to obtain the optimal value of f is to determine it analytically. See section VI for a discussion about this. We present an algorithm that computes the exact optimal minimum frequency for a particular task system τ and a number of *active* processing cores m in $O(n^2 \log_2^2 m)$ time (see Algorithm 2). We then 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 (5). 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 (5) 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 (3). 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.

$$f \geqslant \Psi_\tau(m, \bar{\kappa}(f)). \tag{6}$$

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 (6) which leads to the following definition.

Definition 3 (Minimum optimal frequency)**.** *The minimum optimal frequency of a system τ schedulable on m active 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 (5) holds. Note this minimum corresponds to f when (6) achieves equality.*

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

$$k_i^{-1}(\kappa) \stackrel{\text{def}}{=} \begin{cases} \{f \mid \frac{u_i}{\gamma_{i,\kappa+1}} \leqslant f < \frac{u_i}{\gamma_{i,\kappa}}\} & \text{if } 0 < \kappa \leqslant m \\ [\frac{u_i}{\gamma_{i,1}}, \infty) & \text{otherwise.} \end{cases} \tag{7}$$

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 (6) 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 2 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: $\mathsf{schedulable}(\tau, m, f)$

```
sum ← 0
for τ_i ∈ τ do
    κ_i ← k_i(f)
    if κ_i = m then
        return False
    sum ← sum + κ_i + (u_i − γ_{i,κ_i} × f) / ((γ_{i,κ_i+1} − γ_{i,κ_i}) × f)
return m ⩾ sum
```

Algorithm 2: $\mathsf{minimumOptimalFrequency}(\tau, m)$

```
for i ∈ {1, 2, ..., n} do
    if schedulable(τ, m, u_i/γ_{i,m}) then
        κ̄_i ← m − 1
    else
        κ̄_i ← min_{κ=0}^{m−1} {κ | not schedulable(τ, m, u_i/γ_{i,κ+1})}
κ̄ ≝ ⟨κ̄_1, κ̄_2, ..., κ̄_n⟩
return Ψ_τ(m, κ̄)
```

To compute $\mathsf{schedulable}(\tau, m, f)$, we determine the value of $k_i(f)$ from frequency f according to (3), 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 $O(n \log_2 m)$. See Algorithm 1 for a complete sketch of this algorithm.

Algorithm 3: $\mathsf{frequencyCoreSelection}(\tau, m)$

```
ℓ_min ← 1
f_{ℓ_min} ← minimumOptimalFrequency(τ, 1)
for ℓ ∈ {2, 3, ..., m} do
    f_ℓ ← minimumOptimalFrequency(τ, ℓ)
    if P(f_ℓ, ℓ) < P(f_{ℓ_min}, ℓ_min) then
        ℓ_min ← ℓ
        f_{ℓ_min} ← f_ℓ
return ⟨f_{ℓ_min}, ℓ_min⟩
```

In Algorithm 2 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 2 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 $O(mn^2 \log_2^2 m)$. See Algorithm 3 for the complete description.

B. An Example

Let us use the same example system than previously introduced in Section III-D. Consider $\tau = \{\tau_1, \tau_2\}$ to be scheduled on $m = 3$ identical processors. Tasks are defined as follow : $\tau_1 = (6, 4, \Gamma_1)$ with $\Gamma_1 = (1.0, 1.5, 2.0)$ and $\tau_2 = (3, 4, \Gamma_2)$ with $\Gamma_2 = (1.0, 1.2, 1.3)$. The vector $\bar{\kappa}$ corresponding to this configuration computed by the algorithm is equal to $(\bar{\kappa}_1 = 2, \bar{\kappa}_2 = 0)$. This implies that the optimal minimum frequency (Algorithm 2) for this system to be feasible on 3 processors is equal to $f_{\min}^{(\tau,m)} = \Psi_\tau(3, \langle 2, 0\rangle) = 0.9375$. We can see that if we call the feasibility test function (Algorithm 1) for any frequency greater or equal than 0.9375, it will return True; it will return False for any lower value.

C. Proof of Correctness

Theorem 3. *Algorithm 2 returns $f_{\min}^{(\tau,m)}$.*

Proof Sketch: We only give a brief sketch of the proof. By the arguments in Section V-A, Algorithm 2 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 2. 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 (6) has reached equality with f', this is the smallest frequency such that τ is schedulable. Therefore, f' must be equal to $f_{\min}^{(\tau,m)}$. □

VI. Practical considerations

This section discusses some of our choices and assumptions for this work with regard to reality or practical implementation.

A. Continuous range frequency selection

We made the assumption in section III-C that the homogeneous frequency of the processing platform is drawn from the positive continuous range — *i.e.*, $f \in \mathbb{R}_{>0}$. It is worth mentioning that in practice the available frequencies are always drawn from a discrete and finite set. As this set is reasonably small, exhaustive search (by binary search over all available frequencies) could be an applicable approach to determine the minimum optimal frequency value. This approach has an even better complexity than our analytical approach. However, we choose to provide the exact analytical solution due to the generality of the solution and the fact that continuous frequencies can be emulated with discrete frequencies (as discussed below).

Notice that having the analytical expression of $f_{\min}^{(\tau,m)}$, we can always find the smallest available frequency for which the system remains schedulable. For a given platform with discrete and finite frequency set $\mathbb{F} \stackrel{\text{def}}{=} \{f_1, f_2, \ldots, f_p\}$, with $f_1 < f_2 < \ldots < f_p$, we define the following ceiling operator, which represents the smallest available frequency greater than the given analytical frequency:

$$\lceil f \rceil_{\mathbb{F}} \stackrel{\text{def}}{=} \min\{f_i \in \mathbb{F} \mid f_i \geqslant f\}.$$

We just have to take $\lceil f_{\min}^{(\tau,m)} \rceil_{\mathbb{F}}$ to select the smallest available frequency able to schedule the task system. Notice that if $\lceil f_{\min}^{(\tau,m)} \rceil_{\mathbb{F}} > f_{\min}^{(\tau,m)}$, *i.e.*, the smallest available frequency is slightly higher than the optimal frequency, the system will not be saturated and some idle instants will appear in the execution of the system.

Symmetrically, we can also define a floor operator, which represents the greatest available frequency smaller than the given analytical frequency:

$$\lfloor f \rfloor_{\mathbb{F}} \stackrel{\text{def}}{=} \max\{f_i \in \mathbb{F} \mid f_i \leqslant f\}.$$

Therefore, even if the chosen running platform has only a discrete set of frequencies $\mathbb{F}$, the minimum analytical frequency $f_{\min}^{(\tau,m)}$ could be emulated by switching between the frequency above and the frequency below, removing then the idle instants introduced by taking a too large frequency.

Indeed, the analytical frequency $f_{\min}^{(\tau,m)}$ can be approximated by switching between the two nearest discrete frequencies $f_\ell \stackrel{\text{def}}{=} \lfloor f_{\min}^{(\tau,m)} \rfloor_{\mathbb{F}}$ and $f_h \stackrel{\text{def}}{=} \lceil f_{\min}^{(\tau,m)} \rceil_{\mathbb{F}}$. We have $f \in [f_\ell, f_h]$. Then we define α such that $f_{\min}^{(\tau,m)} = \alpha f_h + (1-\alpha) f_\ell$. Solving this for α will give us the amount of time in each time unit that we should run at the high/low frequencies to obtain $f_{\min}^{(\tau,m)}$, saving then more power than only running the system at frequency f_h, which would have been selected by binary search of the set of frequencies $\mathbb{F}$. Notice that we assume for this that the overheads of switching frequency at run-time can be neglected. Ishihara and Yasuura [19] uses this technique for convex power functions and extends Property 1 for a discrete set of frequencies. This justifies the need for an algorithm computing the exact analytical minimum frequency.

Finally, it is possible that no available frequency is higher than the computed optimal minimum analytical frequency, *i.e.*, $f_{\min}^{(\tau,m)} > f_p = \max(\mathbb{F})$. It means that the malleable jobs of the system cannot meet their deadlines even at the highest speed available on the target platform. It is said so that the system is not schedulable on the chosen platform. Notice that in our simulations, we have made the assumption that it will never be the case: we simply determine $f_{\min}^{(\tau,m)}$ such that the system is schedulable and it requires that the chosen platform is capable of running at this frequency (even if this frequency is greater than 1).

B. Linear dependency between frequency and job execution speed

In section III-C, we made the assumption throughout this paper that there is a linear dependency between processor frequency and execution time. As this is not really accurate in practice – other factors can have huge impact on job execution: cache synchronisation, memory latencies, etc. – this simplifying hypothesis is often made in the scheduling literature, e.g. in the popular *uniform parallel machine* model[1] [20].

To better cope with potential practical implementation, we could define, for each task, a notion of functional utilization:

$$u_i : \mathbb{R}_{>0} \to \mathbb{R}_{>0} : f \mapsto u_i(f)$$

Instead of replacing u_i by $\frac{u_i}{f}$ in the schedulability criteria (as it is done in section IV-A), we would replace u_i by this function $u_i(f)$. The definition of this function would define how the execution time of a task would be impacted by the selection of a given frequency f. As we assumed in the rest of the paper that the dependency between frequency and execution time was linear, we implicitly set $u_i(f) \stackrel{\text{def}}{=} \frac{u_i}{f}$. Other more realistic choices could have been made (and would be in future research) for the definition of $u_i(f)$, but notice that this could have an impact on the optimality of the underlying scheduling algorithm. In particular, for a non-linear dependency, optimality of the canonical schedule would be lost.

It is easy to integrate this non-linear dependency with a restricted and finite set of available frequencies (as discussed in VI-A): we just have to restrict the domain of the function.

VII. Simulations

In order to investigate the potential benefit 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.

[1]Each processor in a uniform parallel machine is characterized by its own computing capacity. A processor with computing capacity s executing a job for t time units is processing $s \times t$ units of execution.

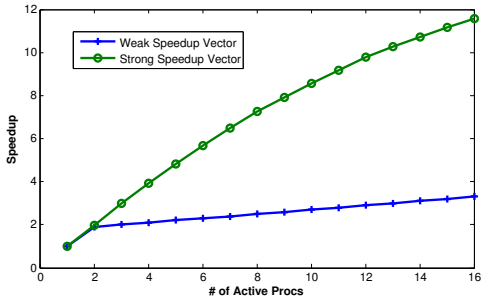


Fig. 2: Speedup Vectors for *strong* and *weak* parallelized systems.

# cores	Γ_{WPS}	Γ_{SPS}	# cores	Γ_{WPS}	Γ_{SPS}
1	1.0	1.000	9	2.6	7.926
2	1.9	1.990	10	2.7	8.559
3	2.0	2.970	11	2.8	9.185
4	2.1	3.913	12	2.9	9.771
5	2.2	4.801	13	3.0	10.271
6	2.3	5.670	14	3.1	10.726
7	2.4	6.505	15	3.2	11.148
8	2.5	7.255	16	3.3	11.558

Fig. 3: Values of speedup vectors for WPS and SPS.

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 [21] that is proven [22] to generate uniformly-distributed multiprocessor task systems (i.e., $u_{\text{sum}} > 1$). As in our case we want to model tasks with individual utilization u_i not bounded by 1,[2] we slightly modified the way the authors of [22] use the `RandomVectorsFixedSum` algorithm to permit $u_i > 1$ when generating task parameters.

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). A task of the SPS will have a speedup vector with better values than the one of the WPS. The values used in our simulations for Γ_{SPS}, the vector for SPS, and Γ_{WPS}, the vector for WPS are presented hereafter. Notice that both vectors respects *sub-linear speedup ratio* and *work-limited parallelism*, as defined in Section III. Simulations are done for 1 to 16 cores, so there is 16 values for each vector. Values of these two vectors are plotted on Figure 2 and explicitly presented in array-format on Figure 3.

[2]With the theoretical interpretation that a task with $u_i > 1$ will either need to be scheduled on a platform with a frequency greater than 1, either be modelled as a malleable tasks and thus scheduled on several processing cores.

3) 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 [23]. The static (leakage) power consumption of a processor can be as high as 42% of total power and depends on many factors [24]. 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, $P(f,k) = f^3k + 0.15 \times k$, where f is the processing frequency and k is the number of active cores; the two additive terms represent dynamic and static power, respectively, and both depend on k. This model is sound regarding the physical behaviour of power consumption on CMOS processors platform, and many former real-time researches such as [23] use it. This model gives us a tool to compare power savings in the parallel schedule w.r.t. to the non-parallel schedule. It is not expressed in *Watts*, as it does not describe a real machine, but is useful to *relatively* compare the different solutions evaluated in our abstract simulations. 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 4.

4) Minimum Frequency/Number of Active Cores Determination: The goal of our simulations is to compare the power consumption for task systems in the three following settings:

- when tasks are strongly parallelized (with SPS vector);
- when tasks are weakly parallelized (with WPS vector);
- when tasks are not parallelized, i.e. scheduled with the traditional non-parallel optimal schedule.

Therefore, for each task system generated in Step 1, we compute three distinct frequency/number of active cores couple values: the two first couple values are those returned by Algorithm 3 to optimally schedule the system when tasks are *strongly* parallelized and *weakly* parallelized. Using this Algorithm we make use of both DVFS (frequency selection) and DPM (number of turned on cores selection). For simplicity in our simulations, when evaluating the minimum frequency to schedule task systems for SPS (resp. WPS), the same vector Γ_{SPS} (resp. Γ_{WPS}) is set to each task of the system. The function $P(f,k)$ used in Algorithm 3 is defined in Step 3. For the third couple value, the frequency/number of active cores selected is the optimal one w.r.t. traditional non-parallel (*i.e.*, sequential) technique. To compute the minimum frequency required for a non-parallel scheduling algorithm (referred to as SEQ) with a fixed number of active cores, remember that the optimal schedulability criteria for sequential multiprocessor system on DVFS platform where the frequency is choosen offline is the following (we assume $\gamma_{i,1} = 1 \, \forall \, i$):

$$\begin{cases} \frac{u_{\max}}{f} \leqslant 1 \\ \frac{u_{\text{sum}}}{f} \leqslant m \, . \end{cases}$$

Therefore, the minimum optimal frequency for a fixed number of cores m is denoted as $f_{\text{seq}}^{(\tau,m)} \stackrel{\text{def}}{=} \max\left(u_{\max}, \frac{u_{\text{sum}}}{m}\right)$. As we want to compare parallel and sequential schedule with the same DVFS/DPM features, we want to select the couple frequency/number of active cores that minimize the function $P(f,k)$ (like in the parallel case). Therefore, we call a modified version of Algorithm 3, with the function

minimumOptimalFrequency(τ, ℓ) returning $f_{\text{seq}}^{(\tau,\ell)}$ instead of $f_{\min}^{(\tau,\ell)}$.

For example, consider $\tau = \{\tau_1, \tau_2\}$ to be scheduled on a platform with $m = 3$ processing cores where $e_1 = 6$, $p_1 = 4$, $e_2 = 3$ and $p_2 = 4$. We then have $u_1 = \frac{3}{2}$ and $u_2 = \frac{3}{4}$ and Algorithm 3 returns, for each of the three settings,

$$\langle f_{\text{sps}}^{(\tau,\ell_{\text{sps}})} = 0.7525, \ell_{\text{sps}} = 3\rangle \mapsto P(3, 0.7525) = 1.72845$$
$$\langle f_{\text{wps}}^{(\tau,\ell_{\text{wps}})} = 0.7875, \ell_{\text{wps}} = 3\rangle \mapsto P(3, 0.7875) = 1.9151$$
$$\langle f_{\text{seq}}^{(\tau,\ell_{\text{seq}})} = 1.5, \ell_{\text{seq}} = 2\rangle \mapsto P(2, 1.5) = 7.05\,.$$

We can already see on this simple example that the gain from sequential to parallel is substantial. Notice that even in non-parallel setting, we allow:

- individual utilization not bounded ($u_i > 1$ is permitted);
- total utilization not bounded ($u_{\text{sum}} > m$ is permitted);
- homogeneous frequency is not bounded ($f > 1$ is permitted).

It is important to notice that utilization is not bounded. For example, it is not a problem to have a total utilization greater than m: it just requires a running homogeneous frequency greater than 1. Like addressed in section VI-A, if the higher frequency available on the target platform is less than the one computed by our algorithm, it means that the task system is not schedulable on this target platform.

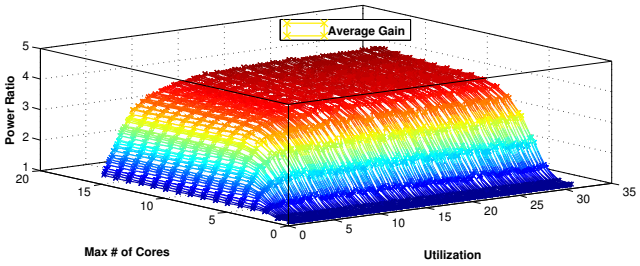
5) Results Comparison: Using the random-task generator introduced in Step 1, we generate task systems with 8 tasks[3]. The total system utilization is varied from 1.5 to 32.0 by 0.1 increments and number of available cores are varied from 1 to 16. The simulation runs for each task system/maximum number of cores pair. For each utilization point, we store the exact frequency and number of active cores returned by Algorithm 3 and the associated power consumed (as defined in Step 3). This is done three times, one for each setting (SPS, WPS and SEQ) as explained in Step 4: 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 4 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 (with the same total utilization).

B. Results & Discussion

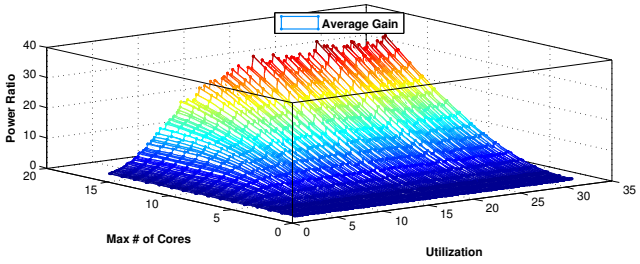
Figures 4a, 4b, and 4c display the power savings obtained from our simulations. Each point represents the average of power saving ratios over 100 randomly generated systems each having a fixed total utilization u_{sum} and maximum m number of processing cores. The *z-values* of the plots are then:

- $\mathbb{E}[\frac{P(f_{\text{seq}}, k_{\text{seq}})}{P(f_{\text{wps}}, k_{\text{wps}})}]$ (Fig. 4a),
- $\mathbb{E}[\frac{P(f_{\text{seq}}, k_{\text{seq}})}{P(f_{\text{sps}}, k_{\text{sps}})}]$ (Fig. 4b),

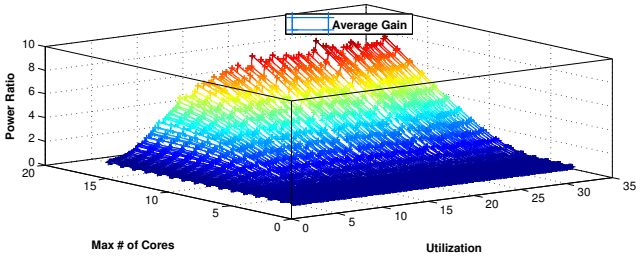
[3]The behaviour for $n \neq 8$ would be quite similar.



(a) Power savings for WPS over SEQ.



(b) Power savings for SPS over SEQ.



(c) Power savings for SPS over WPS.

Fig. 4: Average power savings of parallelized systems.

- and $\mathbb{E}[\frac{P(f_{\text{wps}}, k_{\text{wps}})}{P(f_{\text{sps}}, k_{\text{sps}})}]$ (Fig. 4c),

where, in the setting x, f_{x} represents the minimum optimal frequency and k_{x} represents the number of activated cores ($1 \leqslant k_{\text{x}} \leqslant m$) and $\mathbb{E}[.]$ represents the mean over 100 values.

The figures show then that the proposed algorithm has substantial power savings over sequential optimal algorithm. Furthermore, for strongly parallelized systems, the power saving is substantially larger, it further increases as with the higher system utilization and number of available cores. On the other hand, for weakly parallelized systems, the power gain saturates quickly and does not increases anymore with the system utilization and the number of available cores. We can see on Fig. 4a 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 Fig. 4b).

From these plots, there are a few noticeable trends:

- As the total utilization increases, the power savings increases (for active processors greater than 2); the savings appears to be due to the fact that high utilization workloads require higher frequency in the sequential approach (and thus, more power) and can be easily distributed

amongst cores in the parallel approach.
- As the total number of available cores increase, the power savings increases; the savings appears to be due to the fact that non-parallel schedules will quickly reach the limit where adding core does not impact the schedulability of sequential jobs. Notice that this limits is also reached (less quickly) by the WPS (where it saturates, *cf.* Fig. 4a).

We can conclude from this than the better jobs are parallelized (*i.e.*, the better are the $\gamma_{i,j}$ values of the speedup vector), the better are the power savings.

We can ask if the savings are biased by the fact that u_i can be greater than 1. Indeed, in our framework, in sequential mode of execution, to reach deadlines for task with $u_i > 1$, there is no other choice than increasing the frequency (to a value greater than 1). This could introduce a bias in simulations based on a randomized u_i thant can be either less or greater than 1. However, we can see that power savings are still present when $\forall i : u_i < 1$ with this simple example: $\tau = \{\tau_1, \tau_2\}$, where $e_1 = 1$, $p_1 = 10$, $e_2 = 3$ and $p_2 = 4$. We let the maximum number of cores of the platform be equal to $m = 4$. For this task system, the power values are the following:

$$\langle f_{\text{sps}}^{(\tau,\ell_{\text{sps}})} = 0.4266, \ell_{\text{sps}} = 2\rangle \mapsto P(2, 0.4266) = 0.4553$$
$$\langle f_{\text{wps}}^{(\tau,\ell_{\text{wps}})} = 0.4421, \ell_{\text{wps}} = 2\rangle \mapsto P(2, 0.4421) = 0.4728$$
$$\langle f_{\text{seq}}^{(\tau,\ell_{\text{seq}})} = 0.8500, \ell_{\text{seq}} = 1\rangle \mapsto P(1, 0.8500) = 0.7641$$

By scheduling malleable jobs, we observe in this example relative power savings from 61% to 68% (depending of the tasks' speedup vector). Therefore parallelization helps also for systems with only tasks with $u_i < 1$.

VIII. 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. We can see from simulation results that power savings can be substantial even for system with weak parallelization: they tend to use computational resources more intelligently. For better parallelized system, the power gains can be very high. 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.

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 consider more realistic application models like the *thread* or the *fork-join* model. To avoid over-simplification of platform and power model, we will also consider practical implementations of parallel power-aware online schedulers into a RTOS deployed upon an actual hardware testbed and measure practical power savings directly on this platform.

Acknowledgements

This research has been supported in part by the US National Science Foundation via a CAREER Grant (CNS-0953585), a CSR Grant (CNS-1116787), and a CRI Grant (CNS-1205338), and by Innoviris, the Brussels Regional R&D Funding Agency within the ARTEMIS CRAFTERS project grant Convention 2012 EUART 2a and 2b.

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. London, UK, UK: Springer-Verlag, 1997, pp. 1–34. [Online]. Available: http://dl.acm.org/citation.cfm?id=646378.689517

[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] D. S. Hochbaum and D. B. Shmoys, "A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach," *SIAM journal on computing*, vol. 17, no. 3, pp. 539–551, 1988.

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

[22] 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. 2010, pp. 6–11. [Online]. Available: http://retis.sssup.it/waters2010/waters2010.pdf#page=6

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

[24] 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.