Energy-aware scheduling of bag-of-tasks applications on master-worker platforms

Jean-François Pineau, Yves Robert, Frédéric Vivien

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **00**:1–15 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Energy-aware scheduling of bag-of-tasks applications on master-worker platforms

Jean-François Pineau[*,4] , Yves Robert[1,3] and Frédéric Vivien[2,3]

[1] *ENS Lyon, Université de Lyon and Institut Universitaire de France*
[2] *INRIA and Université de Lyon*
[3] *LIP laboratory, UMR 5668, ENS Lyon–CNRS–INRIA–UCBL, Lyon, France*
[4] *LIRMM laboratory, UMR 5506, CNRS–Université Montpellier 2, France*

SUMMARY

We consider the problem of scheduling an application composed of independent tasks on a fully heterogeneous master-worker platform with communication costs. We introduce a bi-criteria approach aiming at maximizing the throughput of the application while minimizing the energy consumed by participating resources. Assuming arbitrary super-linear power consumption laws, we investigate different models, with energy overheads and memory constraints. Building upon closed-form expressions for the uni-processor case, we derive asymptotically optimal solutions for all models.

KEY WORDS: Energy-aware, overhead, master-worker platforms, bi-criteria, steady-state scheduling

1. Introduction

The Earth Simulator requires about 12 megawatts of peak power, and Petaflop systems may require 100 MW of power, nearly the output of a small power plant (300 MW). At \$100 per Megawatt.Hour, peak operation of a Petaflop machine may thus cost \$10,000 per hour [1]. And these figures ignore the additional cost of dedicated cooling. Current estimates state that cooling costs \$1 to \$3 per watt of heat dissipated [2]. This is just one of the many economical reasons why energy-aware scheduling is an important issue, even without considering battery-powered systems such as laptop and embedded systems.

Many important scheduling problems involve large collections of identical tasks [3, 4]. In this paper, we consider a single bag-of-tasks application which is launched on a heterogeneous platform. We suppose that all processors have a discrete number of speeds (or modes) of

[*]Correspondence to: Jean-François Pineau, LIRMM, 161 rue Ada, 34392 Montpellier, France.
[†]E-mail: pineau@lirmm.fr, {Yves.Robert, Frederic.Vivien}@ens-lyon.fr. This work was supported in part by the ANR StochaGrid project.

computation: the quicker the speed, the less efficient energetically-speaking. Our aim is to maximize the throughput, i.e., the fractional number of tasks processed per time-unit, while minimizing the energy consumed. Unfortunately, the goals of low power consumption and efficient scheduling are contradictory. Indeed, the throughput can be maximized by using more energy to speed up processors, while energy can be minimized by reducing processor speeds, hence the total throughput.

Altogether, power-aware scheduling truly is a bi-criteria optimization problem. A common approach to such problems is to fix a threshold for one objective and to minimize the other. This leads to two interesting questions. If we fix energy, we get the *laptop problem*, which asks "What is the best schedule achievable using a particular energy budget, before battery becomes critically low?". Fixing schedule quality gives the *server problem*, which asks "What is the least energy required to achieve a desired level of performance?".

The major contribution of this work is to consider a fully heterogeneous master-worker platform, and to take communication costs into account. We extend a previous optimal polynomial algorithm that was derived under an ideal energy-consumption model [5] to fully take into account more realistic models. Here is the summary of our main results:

- Under a refined energy-consumption model with overheads, we derive a polynomial algorithm which is asymptotically optimal, i.e. relatively closer to the optimal as the number of processed tasks increase.
- Adding memory constraints to overheads, we consider a model where processor memory is limited. Thus, if the worker runs slower than the desired throughput, it will be forced to switch to a faster mode when the memory will be full. In this context, we determined the best way to minimize the energy consumption while achieving a given throughput on one processor. This represents the first step to adapt our algorithm to this model.

This paper is organized as follows. We first present the framework and different energy consumption models in Section 2. We study the bi-criteria scheduling problem under the model with overheads in Section 3 , and under the more realistic (albeit more difficult) model with memory constraints in Section 4. Section 5 is devoted to an overview of related work. Finally, we state some concluding remarks in Section 6.

2. Framework

We outline the model for the target applications and platforms, as well as the characteristics of the consumption model. Next we formally state the bi-criteria optimization problem.

2.1. Application and platform model

We consider a bag-of-tasks application $\mathcal{A}$, composed of a large number of independent, same-size tasks, to be deployed on a heterogeneous master-worker platform. We let $\boldsymbol{\omega}$ be the amount of computation (expressed in *flops*) required to process a task, and $\boldsymbol{\delta}$ be the volume of data (expressed in *bytes*) to be communicated for each task. We do not consider return messages.

This simplifying hypothesis could be alleviated by considering longer messages (append the return message for a given task to the incoming message of the next one).

The master-worker platform, also called star network, or single-level tree in the literature, is composed of a master P_{master}, the root of the tree, and $\boldsymbol{p}$ workers P_u $(1 \leq u \leq p)$. Without loss of generality, we assume that the master has no processing capability. Otherwise, we can simulate the computations of the master by adding an extra virtual worker paying no communication cost. The link between P_{master} and P_u has a bandwidth $\boldsymbol{b_u}$. We assume a linear cost model: it takes a time δ/b_u to send a task to processor P_u.

We suppose that the master can send/receive data to/from all workers at a given time-step according to the *bounded multi-port* model [6, 7]. There is a limit on the total amount of data that the master can send per time-unit. Intuitively, the bound corresponds to the bandwidth capacity of the master's network card; the flow of data out of the card can be either directed to a single link or split among several links, hence the multi-port hypothesis.

We also assume that computations obey the so-called *synchronous start* computation: the computation of a task on a worker can start at the same time as the reception of the task begins, provided that the computation rate is not greater than the communication rate (the communication must complete before the computation). This models the fact that, in several applications, only the first bytes of data are needed to start executing a task. In addition, the theoretical results of this paper are more easily expressed under this model, which provides an upper bound on the achievable performance. Furthermore, results in [8] show that proofs written under that model can be extended to more realistic models (one-port communication and atomic computation).

2.2. Energy model

Among the main system-level energy-saving techniques, Dynamic Voltage Scaling (DVS) works on a very simple principle: decrease the supply voltage (and hence the clock frequency) to the CPU so as to consume less power. For this reason, DVS is also called *frequency-scaling* or *speed scaling* [9]. We assume a discrete voltage-scaling model. The computational speed of worker P_u has to be picked among a limited number of $\boldsymbol{m_u}$ modes. Computational speeds are denoted as $\boldsymbol{s_{u,i}}$, meaning that processor P_u running in the ith mode (denoted by $P_{u,i}$) needs $\omega/s_{u,i}$ time units to execute one task of $\mathcal{A}$. We suppose that processing speeds are listed in increasing order ($s_{u,1} \leq s_{u,2} \leq \cdots \leq s_{u,m_u}$), and that modes are exclusive: one processor can only run in a single mode at any given time.

Rather than assuming a relation of the form $P_d = s^\alpha$ where P_d is the power dissipation, s the processor speed, and α some constant greater than 1, we adopt a more general approach, as we only assume that power consumption is a *super-linear* function (i.e., above the linear function $f(x) = x$ and convex) of the processor speed. We denote by $\boldsymbol{\mathfrak{P}^{[1]}_{u,i}}$ the instantaneous power consumption (per time unit) of processor $P_{u,i}$.

We focus on the following three energy consumption models. Under the **ideal model**, switching among the modes does not cost any penalty, and an idle processor does not consume any power. Consequently, for each processor P_u, the energy consumption is super-linear from 0 to the power consumption at frequency $s_{u,1}$. In this model, the energy consumption is a linear function of the power consumption ($\mathfrak{P}_{u,i}(t) = \mathfrak{P}^{[1]}_{u,i} \cdot t$). This simpler model will be used in the proofs to get lower bounds on energy consumption.

Under the **model with switching overheads**, the processor pays a consumption penalty at each transition between two modes, this energy overhead depending on the modes; we denote by $\mathfrak{P}_u^{(i \rightarrow j)}$ the energy overhead to switch processor P_u from mode i to mode j. In the literature [10, 11], authors often state that overheads are proportional to the square of the voltage difference between both modes, and they use such a relationship. Instead, in this work we aim at keeping more general assumptions, as long as they are consistent with the super-linearity of power consumption functions. Therefore we assume that overheads are super-linear functions of the difference in power consumption between both modes:

$$\mathfrak{P}_u^{(i \rightarrow j)} = \beta_u \left(\mathfrak{P}^{[1]}_{u,j} - \mathfrak{P}^{[1]}_{u,i}\right) \quad (0 \leq i < j \leq m_u)$$

(with β_u a super-linear function depending on the processor). Furthermore, as $\mathfrak{P}^{[1]}_u$ is super-linear, we know that the following properties hold ($0 \leq i \leq j \leq k \leq l \leq m_u$):

non-decreasing behavior:	$\mathfrak{P}_u^{(j \rightarrow k)} \leq \mathfrak{P}_u^{(i \rightarrow k)}$, and $\mathfrak{P}_u^{(j \rightarrow k)} \leq \mathfrak{P}_u^{(j \rightarrow l)}$
triangular inequality:	$\mathfrak{P}_u^{(i \rightarrow k)} \leq \mathfrak{P}_u^{(i \rightarrow j)} + \mathfrak{P}_u^{(j \rightarrow k)}$,
super-linearity:	$\mathfrak{P}_u^{(i \rightarrow j)} + \mathfrak{P}_u^{(j \rightarrow l)} \geq \mathfrak{P}_u^{(i \rightarrow k)} + \mathfrak{P}_u^{(k \rightarrow l)}$.

Under this more realistic model, energy consumption now depends upon the duration of the interval during which the processor is operating at a given mode, and on the processor's previous mode (the overhead is only paid once during this interval). Under this model, the energy consumption is an affine function of the power consumption:

$$\mathfrak{P}_{u,i}(t) = \mathfrak{P}_u^{(j \rightarrow i)} + \mathfrak{P}^{[1]}_{u,i} \cdot t. \tag{1}$$

We also suppose in this model that there are no memory constraints, and that a processor can receive data while turned off. To understand the last point, one can consider multi-core processors. If at least one core is turned on, then other cores can be turned off and still have some data sent to their memory. This way, cores will have data to process as soon as they are turned on.

Under the last **model with memory constraints**, we suppose that all processors have limited memory, and that they must be turned on to receive any data. This is the most complicated model, but also the most realistic.

2.3. Objective function

Our goal is bi-criteria scheduling: the first objective is to minimize the energy consumption, and the second to maximize the throughput. We decided to solve this bi-criteria problem by bounding one parameter: the throughput. We denote by $\rho_{u,i}$ the throughput of worker $P_{u,i}$ for application $\mathcal{A}$ under a specific schedule, i.e., the average number of tasks the schedule wants P_u to execute using mode i per time-unit. There is a limit to the number of tasks that each processor mode can perform per time-unit. First of all, because $P_{u,i}$ runs at speed $s_{u,i}$, it cannot execute more than $s_{u,i}/\omega$ tasks per time-unit. Second, since P_u can only be at one mode at a time, and given that $\frac{\rho_{u,i}\ \omega}{s_{u,i}}$ represents the fraction of time spent under mode $m_{u,i}$ per time-unit, this constraint can be expressed by:

$$\forall\ u \in [1..p],\ \sum_{i=1}^{m_u} \frac{\rho_{u,i}\ \omega}{s_{u,i}} \leq 1.$$

We add an additional idle mode $P_{u,0}$, whose speed is $s_{u,0} = 0$. As the power consumption per time-unit of $P_{u,i}$, when fully used, is $\mathfrak{P}^{[1]}_{u,i}$ ($\mathfrak{P}^{[1]}_{u,0} = 0$), its power consumption per time-unit with a throughput of $\rho_{u,i}$ is then $\frac{\rho_{u,i}\ \omega}{s_{u,i}} \mathfrak{P}^{[1]}_{u,i}$ (note that we do not take into account the energy overhead needed to get P_u to mode i). We denote by ρ_u the throughput of worker P_u, i.e., the sum of the throughput of each mode of P_u (except the throughput of the idle mode). The total throughput of the platform is denoted by:

$$\rho = \sum_{u=1}^{p} \rho_u = \sum_{u=1}^{p} \sum_{i=1}^{m_u} \rho_{u,i}.$$

We define problem MinPower(ρ) as the problem of minimizing the energy consumption while achieving a throughput ρ. In Section 2.4 we summarize previous results under the ideal model. We extend them to more realistic models in Sections 3 and 4.

2.4. Ideal model

Both bi-criteria problems (maximizing the throughput given an upper bound on energy consumption, and minimizing the energy consumption given a lower bound on throughput) have been studied at the processor level, using particular power consumption laws such as $P_d = s^\alpha$ [12, 13, 14]. We provided an optimal solution to these problems in [15, 5], using the sole assumption that the power consumption is super-linear. A key step is to establish closed-form formulas linking power consumption and throughput on a single processor:

Proposition 1. *Under the ideal energy consumption model, for any processor P_u, the optimal power consumption to achieve a throughput of ρ ($0 < \rho \leq \frac{s_{u,m_u}}{\omega}$) is*

$$\mathfrak{P}_u(\rho) = \max_{0 \leq i < m_u} \left\{ (\omega\rho - s_{u,i}) \frac{\mathfrak{P}^{[1]}_{u,i+1} - \mathfrak{P}^{[1]}_{u,i}}{s_{u,i+1} - s_{u,i}} + \mathfrak{P}^{[1]}_{u,i} \right\},$$

and it is obtained using two consecutive modes, P_{u,i_0} and P_{u,i_0+1}, such that $\frac{s_{u,i_0}}{\omega} < \rho \leq \frac{s_{u,i_0+1}}{\omega}$.

Note that a similar result is given in [15, 5] for the converse problem, namely maximizing the throughout subject to a prescribed bound on power consumption. Thanks to Proposition 1, there is no need to specify the throughput for each frequency on any given processor. One only has to fix a throughput for each processor to know how to achieve the minimum power consumption on that processor. Informally, the power minimization algorithm in [15, 5] sorts processors in non-decreasing order according to their power consumption ratio. This power consumption ratio depends on the different processor modes, and the same processor appears a number of times equal to its number of modes. Formally, the quantities $\left\{ \frac{\mathfrak{P}^{[1]}_{u,i+1} - \mathfrak{P}^{[1]}_{u,i}}{s_{u,i+1} - s_{u,i}} \right\}$ are sorted in non-decreasing order, and the cheapest modes of the processors are selected so that the system can achieve the required throughput. The constraints are that each processor throughput is limited by its maximal frequency, and by the bandwidth of the link between itself and the master. Altogether, this leads to Algorithm 1.

Algorithm 1: Greedy algorithm minimizing power consumption under a given throughput

```
Input: throughput $\rho$ that has to be achieved
for $u = 1$ to $p$ do
    $\mathcal{T}[u] \leftarrow 0$; /* throughput of processor $P_u$ */
$\Phi \leftarrow 0$; /* total throughput of the system */
$\mathcal{L} \leftarrow$ sorted list of the $P_{u_k,i_k}$ such that $\forall\, j,\ \frac{\mathfrak{P}^{[1]}_{u_j,1+i_j} - \mathfrak{P}^{[1]}_{u_j,i_j}}{s_{u_j,1+i_j} - s_{u_j,i_j}} \leq \frac{\mathfrak{P}^{[1]}_{u_{j+1},1+i_{j+1}} - \mathfrak{P}^{[1]}_{u_{j+1},i_{j+1}}}{s_{u_{j+1},1+i_{j+1}} - s_{u_{j+1},i_{j+1}}}$;
while $\Phi < \rho$ do
    $P_{u_k,i_k} \leftarrow \mathrm{next}(\mathcal{L})$; /* selection of next cheapest mode */
    $\rho' \leftarrow \mathcal{T}[u_k]$; /* previous throughput of $P_{u_k}$ (at mode $i_k - 1$) */
    $\mathcal{T}[u_k] \leftarrow \min\left\{\frac{s_{u_k,i_k}}{\omega}; \frac{b_{u_k}}{\delta}; \rho' + (\rho - \Phi)\right\}$; /* new throughput of $P_{u_k}$ (at mode $i_k$) */
    if $\mathcal{T}[u_k] = \frac{b_{u_k}}{\delta}$ then
        $\mathcal{L} \leftarrow \mathcal{L} \backslash \{P_{u_k,j}\}$; /* no need to look at faster modes for $P_{u_k}$ */
    $\Phi \leftarrow \Phi + \mathcal{T}[u_k] - \rho'$;
```

Suppose that the last selected mode in Algorithm 1 is $P_{u_{k_0},i_{k_0}}$. Then:

1. each processor having at least one mode consuming strictly less than $P_{u_{k_0},i_{k_0}}$ is fully used, either at the throughput of the bandwidth if reached (this throughput is achieved according to Proposition 1), or at the largest single fastest mode that consumes strictly less than $P_{u_{k_0},i_{k_0}}$, or at the same mode than $P_{u_{k_0},i_{k_0}}$;

2. any processor whose first non-trivial mode consumes exactly the same as $P_{u_{k_0},i_{k_0}}$ is either not used at all, or fully used at its first non-trivial mode;

3. any processor whose first non-trivial mode consumes strictly more than the mode $P_{u_{k_0},i_{k_0}}$ is not used at all;

4. $P_{u_{k_0},i_{k_0}}$ is used at the minimum throughput such that the system achieves a throughput of ρ (according to Proposition 1).

This algorithm was proven optimal[15, 5] to solve problem MinPower(ρ) under the ideal model, and asymptotically optimal under a model where an overhead is paid each time a processor is turned on. In the next sections, we extend this algorithm and assess its performance under more restrictive models.

3. Model with switching overheads

When we move to more realistic models, the problem gets much more complicated. In this section, we still look at the problem of minimizing the energy consumption of the system with a throughput bound, but now we suppose that there is an energy consumption overhead when switching the mode of a processor. We denote this problem as MinPowerOverhead(ρ). Recall that we do not suppose any memory constraint in this section: the memory of each processor is supposed to be infinite.

To take the switching overhead into account, we suppose that at time 0, P_u is in mode 0, and it must be in mode 0 again at time t. We then define the following behavior on every processor P_u in order to achieve a throughput of ρ_u during t time units; if the throughput is feasible, we know that $\rho_u \leq \min\left\{\frac{s_{u,m_u}}{\omega}; \frac{b_u}{\delta}\right\}$, so $\omega\rho_u$ is between two consecutive modes: $s_{u,i_0} < \omega\rho_u \leq s_{u,i_0+1}$. We would like to use both modes, according to the results of [15, 5] for the ideal energy consumption model: running mode P_{u,i_0} during $t_1 = \frac{t(s_{u,i_0+1} - \rho_u\omega)}{s_{u,i_0+1} - s_{u,i_0}}$ time units, and mode P_{u,i_0+1} during $t_2 = \frac{t(\rho_u\omega - s_{u,i_0})}{s_{u,i_0+1} - s_{u,i_0}}$ time units. But the presence of an overhead when switching from P_{u,i_0} to P_{u,i_0+1} may cost more energy than staying at mode P_{u,i_0+1}: this may happen if $\omega\rho_u$ is very close to s_{u,i_0+1}, or if t is small. Then the best solution may be to use processor P_u in mode i_0+1 during the last $t' = t\left(\frac{\rho_u\omega}{s_{u,i_0+1}}\right)$ time units, after staying at mode 0 (it can still receive data while turned off) during the first $(t-t')$ time unit.

Overall, the energy consumption of P_u during t time units is:

$$\mathfrak{P}_u(t,\rho_u) = \min\left\{\begin{array}{l} \mathfrak{P}_u^{(0\,\to\,i_0)} + \mathfrak{P}_{u,i_0}^{[1]}\cdot t_1 + \mathfrak{P}_u^{(i_0\,\to\,i_0+1)} + \mathfrak{P}_{u,i_0+1}^{[1]}\cdot t_2 + \mathfrak{P}_u^{(i_0+1\,\to\,0)} \\ \mathfrak{P}_u^{(0\,\to\,i_0+1)} + \mathfrak{P}_{u,i_0+1}^{[1]}\cdot t' + \mathfrak{P}_u^{(i_0+1\,\to\,0)} \end{array}\right. \tag{2}$$

Thanks to the properties of overheads (see Section 2.2), we can prove the following result:

Proposition 2. *For any processor P_u, the optimal energy consumption to achieve a throughput of ρ ($0 < \rho \leq \frac{s_{u,m_u}}{\omega}$) during t time units, is obtained either by using the two consecutive modes that surround $\omega\rho$, or by using only the first mode not smaller than $\omega\rho$.*

Proof Let S be an optimal solution to achieve a throughput of ρ during t time units, and let i_0 and i_0+1 be the two consecutive modes that surround $\omega\rho$ ($s_{u,i_0} \leq \omega\rho \leq s_{u,i_0+1}$). Then, we have two cases:

S **uses only one mode:** this mode is greater than (or equal to) i_0+1, otherwise ρ cannot be achieved. Among all possible modes, the overhead to switch P_u from mode 0 to i_0+1 is the smallest (overheads are increasing), and the power consumption at this mode is also the smallest (as $\mathfrak{P}_u^{[1]}$ is super-linear, $t'\mathfrak{P}_{u,i_0+1}^{[1]} = t\rho_u\omega\frac{\mathfrak{P}_{u,i_0+1}^{[1]}}{s_{u,i_0+1}} \leq t\rho_u\omega\frac{\mathfrak{P}_{u,i}^{[1]}}{s_{u,i}}, \forall i \geq i_0+1$).

S **uses more than one mode:** one of these modes $i_{\max}$ has to be greater than (or equal to) i_0+1 (as S is feasible), another ($i_{\min}$) has to be lower or equal to i_0 (otherwise our solution does not consume more energy). Thanks to overhead properties, we know that the total cost of overheads in S, denoted as $\mathfrak{P}_u^{\text{overhead}}$, is:

$$\begin{aligned}
\mathfrak{P}_u^{\text{overhead}} &\geq \mathfrak{P}_u^{(0 \,\to\, i_{\min})} + \mathfrak{P}_u^{(i_{\min} \,\to\, i_{\max})} + \mathfrak{P}_u^{(i_{\max} \,\to\, 0)} \\
&\quad \text{(even if } S \text{ uses more than two modes, due to the triangular inequality)} \\
&\geq \mathfrak{P}_u^{(0 \,\to\, i_{\min})} + \mathfrak{P}_u^{(i_{\min} \,\to\, i_0+1)} + \mathfrak{P}_u^{(i_0+1 \,\to\, 0)} \\
&\quad \text{because } i_{\max} \text{ is greater than (or equal to) } i_0+1 \\
&\geq \mathfrak{P}_u^{(0 \,\to\, i_0)} + \mathfrak{P}_u^{(i_0 \,\to\, i_0+1)} + \mathfrak{P}_u^{(i_0+1 \,\to\, 0)} \quad \text{(due to the super-linearity)}
\end{aligned}$$

Moreover, as $\mathfrak{P}^{[1]}$ is super-linear, we know that the power consumption is smaller when using only modes i_0 and i_0+1 rather than any other (see the proof of Proposition 1 in [15] for details). Overall, our solution does not consume more energy than S, which concludes the proof.

If we rewrite Equation (2) by replacing t_1, t_2, and t' by their values, we can express the energy consumption as a function of the throughput:

$$\mathfrak{P}_u(\rho_u, t) = \min \begin{cases} \max_i \left\{ \left((\rho_u \omega - s_{u,i}) \frac{\mathfrak{P}^{[1]}_{u,i+1} - \mathfrak{P}^{[1]}_{u,i}}{s_{u,i+1} - s_{u,i}} + \mathfrak{P}^{[1]}_{u,i} \right) \cdot t \right. \\ \qquad \left. + \mathfrak{P}_u^{(0 \,\to\, i)} + \mathfrak{P}_u^{(i \,\to\, i+1)} + \mathfrak{P}_u^{(i+1 \,\to\, 0)} \right\} \\ \rho\omega \dfrac{\mathfrak{P}^{[1]}_{u,i_0+1}}{s_{u,i_0+1}} \cdot t + \mathfrak{P}_u^{(0 \,\to\, i_0+1)} + \mathfrak{P}_u^{(i_0+1 \,\to\, 0)} \end{cases} \tag{3}$$

As we are using the same two modes for the same duration as under the ideal model, we see that Equation (3) is very close to the formula described in Proposition 1. We simply add the energy overhead if using these two modes, and we compare it to the energy consumption of the faster mode. Using these new results, we can modify Algorithm 1. The general principle of the approach is as follows: instead of looking at the power consumption per time-unit, we look at the energy consumed during d time units, where d will be defined later. We still take into account the possibility that the worker is bandwidth limited in a separated step.

Algorithm 2: Greedy algorithm minimizing energy consumption under overheads

```
Data: throughput $\rho$ that has to be achieved
Sort in non-decreasing order all the modes of the system $P_{u_k,i_k}$, except those greater than
$\frac{b_{u_k}}{\delta}$, according to the values
    $\dfrac{\left(\mathfrak{P}^{[1]}_{u_j,i_j+1} - \mathfrak{P}^{[1]}_{u_j,i_j}\right) \cdot d + \left(\mathfrak{P}_{u_j}^{(0 \to i_j+1)} + \mathfrak{P}_{u_j}^{(i_j+1 \to 0)} - \mathfrak{P}_{u_j}^{(0 \to i_j)} - \mathfrak{P}_{u_j}^{(i_j \to 0)}\right)}{s_{u_j,i_j+1} - s_{u_j,i_j}}$
Insert one additional mode with a speed of $\omega b_{u_k}/\delta$ per processor
while the throughput $\rho$ is not achieved do
    Select the next cheapest processor
    Increase its throughput to meet its next mode
```

If we consider an ideal model, i.e., where no overheads are paid, then Algorithm 2 behaves like Algorithm 1. Hence, we expect solutions built by Algorithm 2 to become good approximate solutions for problem MinPowerOverhead(ρ) when d becomes large. Indeed we derive the following result:

Theorem 1. *Algorithm 2 is asymptotically optimal for problem* MinPowerOverhead*(ρ).*

Proof If the application $\mathcal{A}$ is composed of $\mathcal{N}$ tasks, the optimal scheduling time will be $\mathcal{T} = \frac{\mathcal{N}}{\rho}$, where ρ is the throughput bound. We denote by $\mathfrak{P}_{opt}$ the optimal energy consumption that would be obtained under the ideal model, $\mathfrak{P}^*$ the optimal energy consumption that can be achieved under the model with switching overheads, and $\mathfrak{P}$ the energy consumption given by Algorithm 2.

As the model with overheads is more constrained than the ideal model, the minimum energy consumption under this model is greater than under the ideal model, so we have $\mathfrak{P}_{opt} \leq \mathfrak{P}^* \leq \mathfrak{P}$. Also, one can remark that, during d time units, the solution given by Algorithm 2 does not consume more energy than the solution given by Algorithm 1 where overhead costs are added.

To prove this last claim, let S be the solution given by Algorithm 2, and S' be the solution given by Algorithm 1. If both algorithms do not select the same modes, because they achieve the exact same throughput, then one processor $P_{i_{\min}}$ has a throughput in S lower than in S' ($\rho_{i_{\min}} \leq \rho'_{i_{\min}}$), and another processor $P_{i_{\max}}$ has a throughput in S greater than in S' ($\rho_{i_{\max}} \geq \rho'_{i_{\max}}$). According to the selection of the modes in Algorithm 2, we know that, during d time units, $P_{i_{\max}}$ consumes no more power at a throughput of $\rho_{i_{\max}}$ than $P_{i_{\min}}$ at a throughput of $\rho'_{i_{\min}}$. So we can build a new solution from S', where $P_{i_{\min}}$ gives a fraction of its work to $P_{i_{\max}}$ until one of these processors reaches its next (or previous) mode. This new solution will consume no more power than S'. If we repeat this exchange pattern, we prove that the solution given by Algorithm 2 consumes no more than the solution given by Algorithm 1.

Next we observe that the solution given by Algorithm 1 under the model with overheads will only pay at most three overheads for each processor, during d time units (two if the processor is in an exact mode, three otherwise). Thus, if we denote by $\mathfrak{P}_u^{[2]}$ the maximum overhead P_u can pay during t time units, we have:

$$\mathfrak{P}(t) \quad \leq \quad \mathfrak{P}_{opt} \cdot t + 3 \cdot \left\lceil \frac{t}{d} \right\rceil \cdot \sum_{u=1}^{p} \mathfrak{P}_u^{[2]} \leq \mathfrak{P}_{opt} \cdot t + 3 \cdot \left\lceil \frac{t}{d} \right\rceil \cdot p \cdot \max_{u=1}^{p} \left\{ \mathfrak{P}_u^{[2]} \right\}.$$

If we fix $d = \sqrt{\mathcal{T}}$, we have

$$\mathfrak{P}(\mathcal{T}) \leq \mathfrak{P}^* \cdot \mathcal{T} + \left(1 + \sqrt{\mathcal{T}}\right) \cdot 3p \cdot \max_{u=1}^{p} \left\{ \mathfrak{P}_u^{[2]} \right\}. \tag{4}$$

Then, we compare $\mathfrak{P}$ and $\mathfrak{P}^*$ during the scheduling of the $\mathcal{N}$ tasks of application $\mathcal{A}$. We use the optimal power consumption times the $\mathcal{T}$ time units as a lower bound of the optimal energy consumption during this $\mathcal{T}$ time units, (i.e., $\mathfrak{P}^*(\mathcal{T}) \geq \mathfrak{P}^* \cdot \mathcal{T}$), and we obtain:

$$\begin{aligned} \frac{\mathfrak{P}(\mathcal{T})}{\mathfrak{P}^*(\mathcal{T})} &\leq 1 + \left(\frac{1}{\mathcal{T}} + \frac{1}{\sqrt{\mathcal{T}}} \right) \frac{3p \cdot \max_{u=1}^{p} \left\{ \mathfrak{P}_u^{[2]} \right\}}{\mathfrak{P}^*} \\ &\leq 1 + \mathcal{O}\left(\frac{1}{\sqrt{\mathcal{T}}} \right). \end{aligned}$$

which achieves the proof of the asymptotic optimality of Algorithm 2.

4. Model with memory constraints

In this section, we consider the model with memory constraints. In addition to paying overheads when switching modes, each processor now has a limited memory, and cannot receive any data whenever turned off. Memory constraints will force each processor to run at a throughput different from one of its own modes, and to switch from its slower mode to its faster mode regularly, when its memory becomes full. Our goal is to provide an optimal single-processor policy for achieving a target throughput ρ: assuming that the data steadily arrives at rate ρ, which modes should be used to minimize energy consumption while meeting the memory constraint?

Let $\mathcal{M}_u$ be the memory bound of P_u. We suppose that at time 0, P_u is at mode 0, with no data in its memory, and it must return into that state at time t.
If the throughput is feasible, we know that $\rho_u \leq \min\left\{\frac{s_{u,m_u}}{\omega}; \frac{b_u}{\delta}\right\}$, and $s_{u,i_0} < \omega\rho_u \leq s_{u,i_0+1}$.
Our strategy is to run the slowest mode P_{u,i_0} first, in order to accumulate data into memory. As δ represents the size of one task, $\rho_u - \frac{s_{u,i_0}}{\omega}$ is the throughput at which the tasks are accumulating into the memory under mode i_0, and $\frac{s_{u,i_0+1}}{\omega} - \rho_u$ is the throughput at which the memory is cleaned under mode $i_0 + 1$. After t'_1 time-units under mode P_{u,i_0}, the memory will be full, and we have to switch to the mode P_{u,i_0+1} during t'_2 time-units until the memory is emptied. We will then repeat this pattern. The values of t'_1 and t'_2 are given by:

$$t'_1 = \frac{\mathcal{M}_u}{\delta\left(\rho_u - \frac{s_{u,i_0}}{\omega}\right)} \quad \text{and} \quad t'_2 = \frac{\mathcal{M}_u}{\delta\left(\frac{s_{u,i_0+1}}{\omega} - \rho_u\right)}.$$

If the memory size is large enough and does not impose any constraint during t times units, or if the remaining time is too small to fill the memory, then t'_1 and t'_2 will be calculated the same way as t_1 and t_2 in Section 3.

As in the previous section, the overhead when switching from P_{u,i_0} to P_{u,i_0+1} may cost more energy than staying at mode P_{u,i_0+1}. Hence we still have to compare the energy consumption of using P_{u,i_0} and P_{u,i_0+1} for the whole t time units and for the computation of last tasks (when there is no enough time to fill the memory) with the energy consumption of staying at mode P_{u,i_0+1} during the corresponding time units.

Overall, the new formula expressing the energy consumption of P_u during t time-units is much more complicated. However, we can prove that the optimal way to achieve a given throughput is to use these two modes exclusively:

Proposition 3. *For any processor P_u with a memory bound of $\mathcal{M}_u$, the optimal energy consumption to achieve a target throughput ρ during t time-units, where ρ also is the data arrival rate and verifies $0 < \rho \leq \frac{s_{u,m_u}}{\omega}$, is obtained either by using the two consecutive modes that surround $\omega\rho$, or by using exclusively the first mode not smaller than $\omega\rho$.*

Proof Let $\mathcal{S}$ be an optimal solution to achieve a throughput ρ during t time-units, and let i_0 and $i_0 + 1$ be the two consecutive modes that surround $\omega\rho$ ($s_{u,i_0} < \omega\rho \leq s_{u,i_0+1}$). Then, we have two cases:

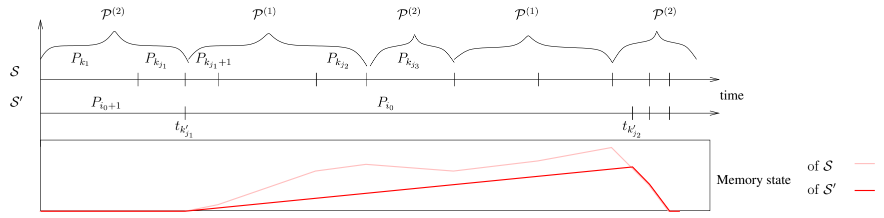

Figure 1. From modes of $\mathcal{P}^{(1)}$ and $\mathcal{P}^{(2)}$ to P_{u,i_0}

• $\mathcal{S}$ **uses only one mode:** this mode is greater than (or equal to) $i_0 + 1$, otherwise ρ cannot be achieved. Among all possible modes, the overhead to switch P_u from mode 0 to $i_0 + 1$ is the smallest (overheads are increasing), and the power consumption at this mode is also the smallest (as $\mathfrak{P}_u^{[1]}$ is super-linear). Furthermore, as the mode speed is greater than the throughput bound, the memory is never used to buffer data for later processing.

• $\mathcal{S}$ **uses more than one mode:** because of the memory bound, $\mathcal{S}$ uses its modes alternatively during the t time-units. Let call $P_{u,k_1}, \cdots, P_{u,k_{\mathcal{K}}}$ the different modes used successively (some may be the same), with P_{u,k_1} used during the first t_{k_1} time-units, P_{u,k_2} used during the next t_{k_2} time-units, and so on. We have $\displaystyle\sum_{j=1}^{\mathcal{K}} t_{k_j} = t$. We split the list of modes used by $\mathcal{S}$ in two, depending on whether the mode is slower or faster than P_{u,i_0}:

$$\mathcal{P}^{(1)} = \left\{P_{u,k_j} | 1 \le j \le \mathcal{K}, k_j \le i_0\right\}, \quad \mathcal{P}^{(2)} = \left\{P_{u,k_j} | 1 \le j \le \mathcal{K}, k_j > i_0\right\}.$$

As $\mathcal{S}$ is feasible, we know that the memory is empty at times 0 and t, and is never overfilled during the whole interval. From that optimal solution, we build another solution, $\mathcal{S}'$.

• We gather all the first modes of $\mathcal{S}$ that belong to $\mathcal{P}^{(2)}$: $P_{u,k_1} \cdots P_{u,k_{j_1}} \in \mathcal{P}^{(2)}$, $P_{u,k_{j_1+1}} \in \mathcal{P}^{(1)}$. As the memory is empty at time 0, and these modes are faster than the throughput bound, we replace them in $\mathcal{S}'$ by P_{u,i_0+1}. The power consumption under this mode is lesser, we pay fewer overheads (only two switching overheads instead of $j_1 + 1$), and the overhead is smaller from P_0 to P_{u,i_0+1} than to P_{u,k_1}, and from P_{u,i_0+1} to $P_{u,k_{j_1+1}}$ than from $P_{u,k_{j_1}}$ (because the first modes of $\mathcal{S}$ are in $\mathcal{P}^{(2)}$ and $P_{u,k_{j_1+1}}$ is in $\mathcal{P}^{(1)}$).

• (Figure 4). From this time $\sum_{j=1}^{k_{j_1}} t_{k_j} = t_{k'_1}$ on, we use P_{u,i_0} in $\mathcal{S}'$ continuously. As the next modes of $\mathcal{S}$ belongs to $\mathcal{P}^{(1)}$, and $\mathcal{S}$ is feasible, and P_{u,i_0} is faster than the modes of $\mathcal{P}^{(1)}$, then the memory is less filled in $\mathcal{S}'$ than in $\mathcal{S}$ until time $\sum_{j=1}^{k_{j_2}} t_{k_j}$. Then, as the memory state function is continuous and upper bounded, and as this function is strictly increasing under $\mathcal{S}'$ (it uses P_{u,i_0} continuously), we know that there exists a time, denoted by $t_{k'_2}$, when the memory state of $\mathcal{S}$ will be the same as the one of $\mathcal{S}'$. At this time, $\mathcal{S}'$ stops to use P_{u,i_0} and uses the same next modes than $\mathcal{S}$. From time $t_{k'_1}$ to $t_{k'_2}$, both schedules have achieved the same amount of work, and they have the same communication rate, and the same memory state at time $t_{k'_1}$ and $t_{k'_2}$), so they have the same average throughput during this interval. This average throughput is achieved under $\mathcal{S}$ by only using P_{u,i_0} continuously, whereas $\mathcal{S}$ uses modes of $\mathcal{P}^{(1)}$ and $\mathcal{P}^{(2)}$. According to our previous work on the ideal model [15], we know that the power consumption of $\mathcal{S}'$ is then smaller than one of $\mathcal{S}$. Furthermore, $\mathcal{S}'$ pays less overheads than $\mathcal{S}$ (only two, instead of $k_{j_2} - k_{j_1} + 1$ in $\mathcal{S}$), and smaller overheads (thanks to the non-decreasing behavior of the overhead function). We repeat this pattern for all sets of modes of $\mathcal{P}^{(1)}$ in $\mathcal{S}$.

• At this step, $\mathcal{S}'$ is only composed of P_{u,i_0} and modes of $\mathcal{P}^{(2)}$. The next step is to build another schedule $\mathcal{S}''$ which splits the work of modes of $\mathcal{P}^{(2)}$ among P_{u,i_0} and P_{u,i_0+1}. If we look at the memory state backwards, we face a similar case as previously; because the memory state at the end of the t time units is the same for both schedules, and since P_{u,i_0+1} cleans the memory slower than modes of $\mathcal{P}^{(2)}$, then the memory is less filled under $\mathcal{S}''$ than under $\mathcal{S}'$. So there exists a time when both schedule had the same memory state, which defines the time window to use P_{u,i_0+1} in $\mathcal{S}''$. During this time window, $\mathcal{S}''$ does not consume more energy than $\mathcal{S}'$, so we can replace each mode of $\mathcal{P}^{(2)}$ in $\mathcal{S}'$ by P_{u,i_0+1} in $\mathcal{S}''$.

• Now $\mathcal{S}''$ is an optimal schedule that uses only P_{u,i_0} and P_{u,i_0+1}. We will gather the utilization of P_{u,i_0} and P_{u,i_0+1} to fully fill and empty the memory, starting with P_{u,i_0}. The power consumption will not change, because the modes will be used the same average time, and the number of switching overheads will not increase. Overall, this scheduling (which is the one defined at the beginning of the section) is optimal.

Altogether, Proposition 3 only is a first step towards designing an efficient schedule for a master-worker platform that obeys the fully realistic model that is dealt with in this section. Indeed, it remains to decide which fraction of the total load should be assigned to each worker, and at which throughput. But Proposition 3 provides the optimum solution for each worker to execute its prescribed share of the work.

5. Related Work

Several papers have been targeting the minimization of power consumption. Most of them suppose that processors can switch to arbitrary speed values.

• **Unit time tasks.** Bunde [14] focuses on the problem of offline scheduling unit time tasks with release dates, while minimizing the makespan or the total flow time on one processor. He extends his work from one processor to multi-processors, but unlike this paper, does not take any communication time into account. He also proves the NP-completeness of the problem of minimizing the makespan on multi-processors with jobs of different amount of work. Authors in [12] concentrate on minimizing the total flow time of unit time jobs with release dates on one processor. After proving that no online algorithm can achieve a constant competitive ratio if job have arbitrary sizes, they exhibit a constant competitive online algorithm and solve the offline problem in polynomial time. Contrarily to [14] where tasks are gathered into blocks and scheduled with increasing speed in order to minimize the makespan, here the authors prove that the speed of the blocks needs to be decreasing in order to minimize both total flow time and energy consumption. Bansal et al [16] improved a part of this work and gave algorithms for the more general problem with arbitrary jobs.

• **Communication-aware.** In [17], the authors are interested about scheduling task graphs with data dependencies while minimizing the energy consumption of both the processors and the inter-processor communication devices. They demonstrate that in the context of multiprocessor systems, inter-processor communications are an important source of energy consumption, and their algorithm reduces up to 80% of the communications. They focus on multiprocessor problems, and consider the energy consumption of the communications, while assuming the communication times negligible compared to the computation times.

• **Discrete voltage case.** In [18], the authors deal with the problem of scheduling tasks on a single processor with discrete voltages. They also look at the model where the energy consumption is related to the task, and describe how to split the voltage for each task. They extend their work in [19] to online problems. In [20], the authors add the constraint that the voltage can only be changed at each cycle of every task, in order to limit the number of transitions and thus the energy overhead. They find that under this model, the minimal number of frequency transitions in order to minimize the energy may be greater than two.

• **Deadlines.** In [21], the authors focus on the problem where tasks arrive according to some release dates. They show that during any elementary time interval defined by some release dates and deadlines of applications, the optimal voltage is constant, and they determine this voltage, as well as the minimum constant speed for each job. [13] improves the best known competitive ratio to minimize the energy while respecting all deadlines. [22] works with an overloaded processor (which means that no algorithm can finish all the jobs) and tries to maximize the throughput. Their online algorithm is $O(1)$ competitive for both throughput maximization and energy minimization. [23] has a similar approach by allowing task rejection, and proves the NP-hardness of the problem under study.

• **Slack sharing.** In [24, 25], the authors investigate dynamic scheduling. They consider the problem of scheduling DAGs before deadlines, using a semi-clairvoyant model. For each task, the only information available is the worst-case execution time. Their algorithm operates in two steps: first a greedy static algorithm schedules the tasks on the processors according to their worst-case execution times and the deadline, and reduces the processor speeds so that each processor meets the deadline. Then, if a task ends sooner than according to the static algorithm, a dynamic slack sharing algorithm uses the extra-time to reduce the speed of computations for the following tasks. However, they do not take communications into account.

• **Heterogeneous multiprocessor systems.** Authors in [26] study the problem of scheduling real-time tasks on two heterogeneous processors. They provide a FPTAS to derive a solution very close to the optimal energy consumption with a reasonable complexity. In [27], the authors propose a greedy algorithm based on affinity to assign frame-based real-time tasks, and then they re-assign them in pseudo-polynomial time when any processing speed can be assigned for a processor. Authors of [28] propose an algorithm based on integer linear programming to minimize the energy consumption without guarantees on the schedulability of a derived solution for systems with discrete voltage. Finally, [29, 30] explored the search of approximation algorithms to minimize processor allocation cost under energy constraints.

• **Overheads.** In [10], the authors consider the scheduling of DAGs, while optimizing both dynamic power consumption and leakage power consumption. Their algorithm trades energy consumption and task execution time. In [11], the authors also consider the execution of tasks with dependencies. They take transition overheads into account, and prove complexity results:

the discrete voltage scaling problems, with and without overheads (as in Sections 3 and 2.4), are strongly NP-hard, while continuous voltage scaling can be solved in polynomial time.

6. Conclusion

In this paper, we have dealt with the problem of scheduling a bag-of-tasks application on a heterogeneous master-worker platform, with minimal energy consumption. Our study takes transition overheads and memory constraints into account. Starting from an optimal algorithm under an ideal model without overhead nor memory constraint, we were able to provide an asymptotically optimal solution for the former extension, with transition overheads. Adding memory constraint, we have derived an important property characterizing how to efficiently achieve a target throughput. We hope that our results will provide a sound theoretical basis for forthcoming studies.

As future work, it would be interesting to study the absolute performance of our algorithm under the new models, in order to assess whether its performance can be guaranteed within a given bound from the optimal. It would also be interesting to test our algorithm through some simulations and actual experiments. We intend to use the Grid5000 testbed [31] as we have on-line access to the electrical consumption of the clusters. Even if individual consumption of the nodes is not available (only groups of nodes), we hope to see if there are any major difference between our model forecasts and the actual consumption of the nodes.

REFERENCES

1. Ge R, Feng X, Cameron KW. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*, IEEE Computer Society: Washington, DC, USA, 2005; 34, doi:http://dx.doi.org/10.1109/SC.2005.57.
2. Skadron K, Stan MR, Sankaranarayanan K, Huang W, Velusamy S, Tarjan D. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization* 2004; **1**(1):94–125, doi:http://doi.acm.org/10.1145/980152.980157.
3. Casanova H, Berman F. *Grid Computing: Making The Global Infrastructure a Reality*, chap. Parameter Sweeps on the Grid with APST. John Wiley, 2003. Hey, A. and Berman, F. and Fox, G., editors.
4. Adler M, Gong Y, Rosenberg AL. Optimal sharing of bags of tasks in heterogeneous clusters. *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, ACM Press, 2003; 1–10.
5. Pineau JF, Robert Y, Vivien F. Energy-aware scheduling of flow applications on master-worker platforms. *Proceedings of Euro-Par 2009*, *LNCS*, vol. 5704, Springer Verlag, 2009; 281–292.
6. Hong B, Prasanna V. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. *International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE Computer Society Press, 2004.
7. Hong B, Prasanna VK. Adaptive allocation of independent tasks to maximize throughput. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 2007; **18**(10):1420–1435.
8. Pineau JF. Communication-aware scheduling on heterogeneous master-worker platforms. PhD Thesis, ENS Lyon 2008. Available at `http://graal.ens-lyon.fr/~jfpineau/pubs/thesis_jfpineau.pdf`.
9. Hotta Y, Sato M, Kimura H, Matsuoka S, Boku T, Takahashi D. Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster. *International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE Computer Society Press, 2006.
10. Martin SM, Flautner K, Mudge T, Blaauw D. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, ACM: New York, NY, USA, 2002; 721–725, doi:http://doi.acm.org/10.1145/774572.774678.

11. Andrei A, Schmitz M, Eles P, Peng Z, Al-Hashimi BM. Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, IEEE Computer Society: Washington, DC, USA, 2004; 10 518.
12. Albers S, Fujiwara H. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms* 2007; **3**(4):49.
13. Bansal N, Kimbrel T, Pruhs K. Dynamic speed scaling to manage energy and temperature. *International Journal of Foundations of Computer Science (IJFCS)* 17-19 Oct 2004; :520–529.
14. Bunde DP. Power-aware scheduling for makespan and flow. *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, ACM Press: New York, NY, USA, 2006; 190–196.
15. Pineau JF, Robert Y, Vivien F. Energy-aware scheduling of flow applications on master-worker platforms. *Research report*, LIP, ENS Lyon October 2008.
16. Bansal N, Pruhs K, Stein C. Speed scaling for weighted flow time. *Proceedings of the ACM-SIAM Symposium On Discrete Algorithms (SODA)*, Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2007; 805–813.
17. Varatkar G, Marculescu R. Communication-aware task scheduling and voltage selection for total systems energy minimization. *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE Computer Society Press: Washington, DC, USA, 2003; 510, doi:http://dx.doi.org/10.1109/ICCAD.2003.51.
18. Ishihara T, Yasuura H. Voltage scheduling problem for dynamically variable voltage processors. *International Symposium on Low Power Electronics and Design (ISLPED)*, ACM Press: New York, NY, USA, 1998; 197–202, doi:http://doi.acm.org/10.1145/280756.280894.
19. Okuma T, Ishihara T, Yasuura H. Real-time task scheduling for a variable voltage processor. *International Symposium on System Synthesis (ISSS)*, IEEE Computer Society Press: Washington, DC, USA, 1999; 24.
20. Zhang Y, Hu XS, Chen DZ. Energy minimization of real-time tasks on variable voltage processors with transition energy overhead. *Asia South Pacific Design Automation Conference (ASPDAC)*, ACM Press: New York, NY, USA, 2003; 65–70, doi:http://doi.acm.org/10.1145/1119772.1119786.
21. Quan G, Hu X. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. *Design Automation Conference*, 2001; 828–833.
22. Chan HL, Chan WT, Lam TW, Lee LK, Mak KS, Wong PWH. Energy efficient online deadline scheduling. *Proceedings of the ACM-SIAM Symposium On Discrete Algorithms (SODA)*, Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2007; 795–804.
23. Chen JJ, Kuo TW, Yang CL, King KJ. Energy-efficient real-time task scheduling with task rejection. *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, European Design and Automation Association: San Jose, CA, USA, 2007; 1629–1634.
24. Zhu D, Melhem R, Childers BR. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 2003; **14**(7):686–700.
25. Rusu C, Melhem R, Mossé D. Multi-version scheduling in rechargeable energy-aware real-time systems. *Journal of Embedded Computing* 2005; **1**(2):271–283.
26. Chen JJ, Thiele L. Energy-efficient task partition for periodic real-time tasks on platforms with dual processing elements. *International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE Computer Society Press, 2008.
27. Huang TY, Tsai YC, Chu EH. A near-optimal solution for the heterogeneous multi-processor single-level voltage setup problem. *International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE Computer Society Press, 2007.
28. Yu Y, Prasanna V. Power-aware resource allocation for independent tasks in heterogeneous real-time systems. *International Conference on Parallel and Distributed Systems (ICPADS)* Dec 2002; :341–348doi: 10.1109/ICPADS.2002.1183422.
29. Chen JJ, Kuo TW. Allocation cost minimization for periodic hard real-time tasks in energy-constrained dvs systems. *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, ACM, 2006; 255–260.
30. Hsu HR, Chen JJ, Kuo TW. Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint. *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, European Design and Automation Association, 2006; 1061–1066.
31. Cappello F, Desprez F, Dayde M, Jeannot E, Jegou Y, Lanteri S, Melab N, Namyst R, Primet P, Richard O, *et al.*. Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform. *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (Grid'2005)*, IEEE Computer Society Press, 2005.