# LEneS: Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processors

Flavius Gruian

Department of Computer Science
Lund University
Box 118, SE-221 00 Lund, Sweden
Tel: +46 46 222 0000
Fax: +46 46 13 1021
e-mail: Flavius.Gruian@cs.lth.se

Krzysztof Kuchcinski

Department of Computer Science
Lund University
Box 118, SE-221 00 Lund, Sweden
Tel: +46 46 222 3414
Fax: +46 46 13 1021
e-mail: Krzysztof.Kuchcinski@cs.lth.se

**Abstract — The work presented in this paper addresses minimization of the energy consumption of a system during system-level design. The paper focuses on scheduling techniques for architectures containing variable supply voltage processors, running dependent tasks. We introduce our new approach for Low-Energy Scheduling (LEneS) and compare it to two other scheduling methods. LEneS is based on a list-scheduling heuristic with dynamic recalculation of priorities, and assumes a given allocation and assignment of tasks to processors. Our approach minimizes the energy by choosing the best combination of supply voltages for each task running on its processor. The set of experiments we present shows that, using the LEneS approach, we can achieve up to 28% energy savings for the tightest deadlines, and up to 77% energy savings when these deadlines are relaxed by 50%.**

## I. Introduction

Mobile computing and communication require careful design from the energy consumption point of view, since battery life-span plays an essential role. Targeting low energy and low power as early as possible in the design process, at high levels of abstraction, is most prolific [1]. Although there has been much work in the area of behavioral synthesis for multiple supply voltages (e.g. [11]), the problem of scheduling supply voltages at system level is different in principle. At behavioral level, once a unique supply voltage for a functional unit is determined during design, it will be constant at runtime. At system level, the supply voltage of a dynamic voltage processor can vary at runtime, which offers more flexibility and better potential for reducing the energy. For this reason, several researchers have addressed energy issues as specific problems at system level [1-9]. Selecting the system architecture and the distribution of the computations can greatly influence the overall energy consumption [5,8,9]. Although shutting down idling parts is one way to further reduce the energy consumption [2,3], it is more effective to slow down selected computations and run the processing units at lower supply voltages [6]. The most worthwhile configuration to decrease the energy consumption is to have processors able to change their supply voltage and frequency during execution. Processors operating at a discrete range of supply voltages have already been designed [12,13]. An optimal preemptive scheduling algorithm for independent tasks running on a single processor with variable speed is described in [4]. In [6], the authors present a low-energy oriented heuristic for non-preemptive scheduling of independent tasks, on a single processor core with variable supply voltage. In [7], Ishihara and Yasuura present another scheduling technique, employing variable voltage processors and independent tasks. There, the authors show that the energy can be further reduced by allowing each task to run in two phases, at two different supply voltages. In our work, we address non-preemptive scheduling on processors with variable supply voltage, overcoming some of the previous limitations. Namely, we propose here an algorithm which handles dependent tasks distributed over several processors.

The rest of the paper is organized as follows. Section II briefly describes the low-energy design flow we have adopted. In section III, we give the relation between energy and voltage followed by an illustrative example. Section IV summarizes three scheduling alternatives for low energy, including our new approach. Section V describes how we model the problem, and section VI presents our scheduling approach. Section VII contains the experimental results, and the conclusions are gathered in section VIII.

## II. Design Flow Overview

We assume a target architecture composed of several dynamic supply voltage processors, with local memories, connected through buses or/and point to point links. The design is modeled as a task-graph, each task being executed on one of the processors. The total execution time for the task-graph is constrained by a designer imposed deadline. The design flow we consider is a step-by-step process, as depicted in Fig. 1. First, the processors are allocated and the tasks are assigned to processors. These two steps can be performed either by the

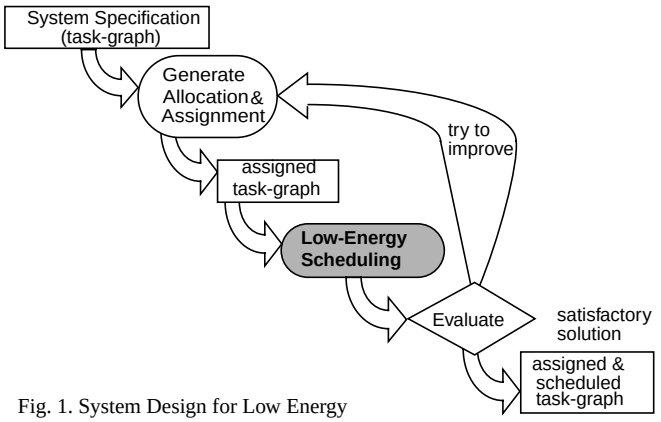


Fig. 1. System Design for Low Energy

designer or by a design tool. Next, an enhanced task-graph (see section V) undergoes a low-energy scheduling step, complying to the already decided allocation and assignment. Then, the solution is evaluated. If the purpose is design space exploration, all these steps must take little time, to allow fast covering of as many options as possible. A more detailed description of our view of a low-energy directed design flow is given in [10]. The work presented in this paper is focused on the scheduling step, for which we developed a heuristic suitable for the described design flow.

## III. The Supply Voltage and the Energy

Consider that task *T* is executing during *N* clock cycles on a processor *P*, which runs at supply voltage *V* and frequency *f*. For the given voltage *V*, processor *P* will have an average power consumption $\pi$. The energy consumed by executing task *T* on processor *P*, running at supply voltage *V*, is computed as: $E = (N \cdot \pi)/f$. The average power consumption dependency on the supply voltage and execution frequency is given by: $\pi = K_a \cdot f \cdot V^2$, where $K_a$ is a task/processor dependent factor, determined by the switched capacitance. Combining the above two formulae, we can rewrite the energy expression as: $E = N \cdot K_a \cdot V^2$. From this, we conclude that lowering the supply voltage would yield a drastic decrease in energy consumption. At the same time, the supply voltage affects the circuit delay, which sets the clock frequency, since $f \sim 1/delay$. Formally, $delay = K_b \cdot V/(V - V_T)^2$, where $V$ is the supply voltage, $V_T$ is the threshold voltage, and $K_b$ is a constant. Thus, decreasing the voltage leads to lower clock frequencies and to longer execution delays.

Processors operating at a range of voltages and frequencies are under development [12]. They are able to adjust their supply voltage, using a fine step, according to the required operating frequency. Processors supporting several supply voltages are already available. For such processors, the various voltages yield different execution delays and energy consumption for the same task *T*. When the processor is limited to a discrete number of supply voltages, the most energy-efficient scenario is to split the task *T* into parts, which will execute at different available voltages. Fig. 2 depicts an illustrative example of scheduling a task-graph (a). In a classic scheduling

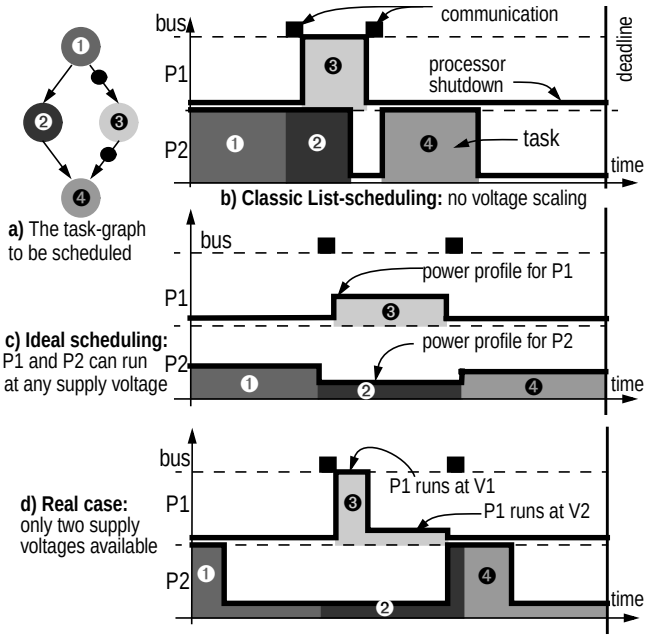


Fig. 2. Scheduling for low energy on multiple voltage processors. The area under the power profile reflects the energy consumption.

technique, such as list-scheduling, voltage selection is not considered (b). In order to obtain energy savings, one must choose the best supply voltage for each processor, depending on the executing task (c). In practice, this can not be done, since the number of available supply voltages for a processor is limited. In (d) a feasible schedule is depicted, using processors with two voltages.

In the case of real processors, with a limited number of supply voltages, the minimal energy for executing a task is obtained by using only two different supply voltages, as shown in [7]. These voltages are the ones around the "ideal" supply voltage for the given deadline. The "ideal" supply voltage is the unique voltage which sets the execution time for that task exactly to the deadline. Only "ideal" processors, with a continuous range of supply voltages, can use an "ideal" voltage. When the allowed task execution time *t* is between $t_1$ (obtained for $V_1$) and $t_2$ (obtained for $V_2$), the task will execute partly at $V_1$, and partly at $V_2$. The execution time *t* can be expressed depending on the number of processor cycles run at the two supply voltages, for the resulting clock frequencies, $f_1$ and $f_2$: $t = xf_1 + (N - x)f_2$, where *x* is the number of cycles run at supply voltage $V_1$. Moreover, task energy can be expressed as a function of clock cycles: $E(x) = K_a(xV_1^2 + (N - x)V_2^2)$. From the last two equations we can deduce the expression of energy as a function of time, *E(t)*, which is a linear dependency. A certain execution duration for a task uniquely identifies the number of cycles needed to execute at the different voltages.

The model described above (depicted in Fig. 2.d) is the execution model assumed in our approach. A processor executes a task in two phases, at two different supply voltages, chosen from the available voltages. Whenever the processor is idle, it is shutdown. In this case, the ideal energy function is approximated by segments between the neighboring voltages (see Fig. 4). A larger number of available voltages yield a better approx-

imation of the ideal energy function, leading to a more efficient design.

## IV. Scheduling Alternatives for Low Energy

There are a number of methods for scheduling the execution of tasks in order to fulfill the deadline and have a minimal energy consumption at the same time. Task dependencies and restrictions imposed by processor implementation complicate the problem. In this section, we summarize several scheduling strategies, applicable in different situations. The last of them is the new scheduling method we introduce in this paper.

### *A. Ideal Case*

Consider first the ideal case, when the tasks are independent, running on one processor. The processor is also ideal, in the sense that it can run at any supply voltage and the threshold voltage is always small enough to be negligible. With all these assumptions, the energy-optimal schedule can be directly obtained from the shortest possible schedule by scaling. The scaling factor is the ratio between the desired deadline and shortest possible deadline (see Appendix for proof). The new execution delays of the tasks can be associated with a new supply voltage. Note that the processor will run at the same, single supply voltage for all tasks. This method can be directly extended to several processors executing independent tasks.

### *B. List-Scheduling with Scaling*

One of the simplifying assumptions for the ideal case was "independent tasks". One can overcome this by using a scheduling strategy that can handle partial ordered tasks or task-graphs. Therefore, we consider list-scheduling with critical path as a priority function (later on referred to as ClassicLS). The scheduling strategy is similar to the one described in subsection A. The task graph is scheduled using list-scheduling, obtaining the tightest possible deadline. Then, the schedule is scaled to fit the desired deadline. The scaling factor is the ratio between the desired deadline and the deadline obtained by list-scheduling. We call this method ScaledLS. The main difference between this approach and the ideal case arises when one considers several processors. In the ideal case every processor had its own scaling factor, while in this case there is a unique scaling factor, given by the overall deadline.

Using this strategy, the tasks on the critical path are scheduled in an optimal way, as in the ideal case. The drawback of this approach is that the tasks on the non-critical paths do not take advantage of the available time slack. Yet, this can be solved by using an appropriate priority function, as described in this paper.

### *C. Our Approach to Low-Energy Scheduling*

The Low-Energy Scheduling algorithm (LEneS) presented in this paper is based on list-scheduling with an energy-sensitive priority function. The priority function is constructed in such way that it handles real processors, able to run at few different supply voltages. The other important feature of our priority function is that it takes advantage of the time slack from the non-critical path. Thus, even for the tightest deadline, there are time moments when the processors run at lower supply voltage, saving energy. For more loose deadlines, we assume that we use the same strategy as in the previous methods: scaling of the tightest schedule (ScaledLEneS).

## V. The Enhanced Task-Graph

The LEneS algorithm works on enhanced task-graphs (ETG), which is a data structure derived from task-graphs. Fig. 3.a) depicts an assigned task-graph. The tasks are represented by the circles annotated by pairs of values. A pair consists of the execution time of the task on the specific processor, running at the reference supply voltage, and the identifier of the processor executing the task. The black disks represent communications annotated with the duration and the identifier of the bus/link used for that specific communication. The arcs define the partial order of task execution, which is imposed by the various data dependencies. The assigned task-graph, which is the output of the allocation and assignment steps (Fig. 1), is transformed into an ETG used by the LEneS algorithm. Fig. 3.b) depicts an example of an ETG. The ETG is obtained from the initial task-graph by substituting each node with a pair of nodes: a start node (the circles), marking the beginning of the execution of that node, and an end node (the grey disks), marking the completion of the task. The execution times of the tasks are now assigned to the internal edges. In our current model, only computational tasks are subject to change their execution delay, while the communication delays remain fixed. The thick edges in the ETG represent the fixed delays. The other edges depict modifiable delays, and the associated numbers define their minimal values. The information regarding the assignment of tasks to processors is also transferred to the ETG.

During scheduling with LEneS, each ETG node will be assigned a time moment, such that the partial order and assignment are respected while the delays and the deadline are not violated, and the energy consumption is minimized.

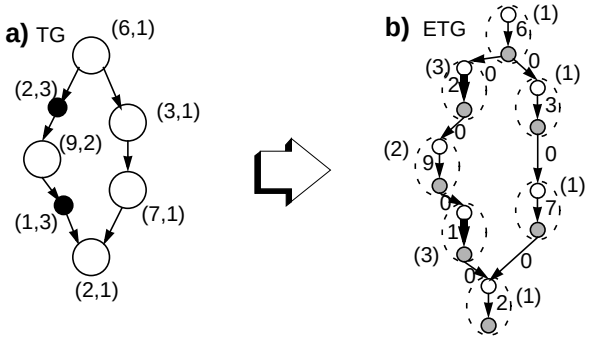


Fig. 3. A task-graph (a) is transformed into an enhanced task-graph (b)

## VI. The LEneS Approach

LEneS, our low-energy scheduling approach (Fig. 5), is based on list-scheduling with a energy sensitive priority function. In every scheduling step, the node priorities change and have to be recalculated. Moreover, the priority function is tuned during several scheduling attempts. Whenever a scheduling attempt fails (the deadline is violated), the priority function is adjusted and a re-scheduling is attempted.

Next, we give the background necessary for understanding the priority function, followed by the expression of the priority function used in our scheduling algorithm. Finally, we present the method used for tuning the priority function.

### A. The Energy of a Schedule

In the ETG, each task is described by the start-end node pair. By performing As-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) schedules on the ETG, for the highest supply voltage, we obtain the ASAP and ALAP time slots, for both start and end nodes. Without any resource constraints, the execution delay of the task can be anywhere between the shortest possible ($t_0$, determined by the highest supply voltage) and the limit imposed by the dependencies in the task-graph ($ALAP_{end}$-$ASAP_{start}$). The energy consumed by the task in this situation is approximated by a linear dependency on its execution time (Fig. 4), as discussed in section III. Given the energy function *E(t)* (section III), we define the *average energy of an ETG node*. We consider that the start-nodes have zero energy, while the average energy of the end-nodes model the task energy. Given that an end-node can be scheduled anywhere in a certain time interval [*a*,*b*], included in its ASAP-ALAP interval, its average energy is:

$$\overline{E_{[a,b]}} = \frac{1}{b-a}\cdot\int_a^b E(t)dt,\ ASAP \le a < b \le ALAP \quad \text{(Fig. 4)}$$

We consider the average energy over an interval [*a, b*] for a certain end-node, as a measure of the quality of the set of solutions obtained by scheduling that end-node in [*a, b*]. At limit, $\overline{E_{[c,c]}} = E(c)$ is the energy yielded by scheduling that node exactly at moment *c*. For a given node, we are able to compare different possible time intervals, or sets of solutions, using the average energy as a measure.

The notion of average energy can be extended to sets of schedules of an ETG. We consider that *a schedule* is an assignment of time moments to all its *N* nodes, such that the partial order is preserved and the given deadline is met. We define a *partial schedule of an ETG* as the set of all possible schedules, given that a certain node can be scheduled anywhere inside a certain time interval. More formal, a partial schedule is an assignment of an interval $[a_i, b_i]$ to each node *i*, where $ASAP_i \le a_i \le b_i \le ALAP_i$, and $\forall k \in 1\ldots N \wedge t_k \in [a_k,b_k]$, $\exists\{t_j | j \in 1\ldots N, j \ne k, t_j \in [a_j,b_j]\}$ such that $\{t_i | i \in 1\ldots N\}$ is a schedule. We define *the average energy of a partial schedule* using the average energy of a node as:

$$\bar{E} = \sum_{i=1}^{N} \overline{E_{[a_i, b_i]}}$$

Since all the start-nodes are considered to have zero energy, the sum given above involves only the end-nodes. Given two partial schedules, we consider that one is better than the other if it has lower average energy.

We say that a partial schedule $S^1$ *covers* another partial schedule $S^2$ if $\forall i, a_i^1 \le a_i^2 \wedge b_i^2 \le b_i^1$, where $a_i$ and $b_i$ have the same meaning as in the previous paragraph, and the superscripts identify the partial schedule. In this case, $S^2$ is a subset of the set of schedules represented by $S^1$.

The idea of the LEneS algorithm consists in choosing, in each scheduling step, a partial schedule covered by the old one. The chosen partial schedule should have the smallest possible energy compared to all the other partial schedules. The starting partial schedule must be one covering for sure all the possible schedules. Therefore, for the initial partial schedule we use the [ALAP, ASAP] intervals without resource constraints.

### B. The Priority Function

The priority function for a node reflects the energy gain (or loss) induced by a specific scheduling decision. At a certain time step *t*, there are nodes (with index *i)* which are eligible for scheduling. If they are delayed, their time intervals will change from $[t, e_i]$ to $[t+1, e_i]$. This change can propagate to the nodes ordered after them. For each of the eligible nodes at a certain scheduling step, delaying the node with one time unit gives the new partial schedule, with a corespondent average energy, $\overline{E^i_{t+1}}$. We are interested in the partial schedule yielding the largest energy reduction. Therefore, our priority function for a node *i*, about to be scheduled at a certain time step *t*, is computed as the difference between the average energy of the current partial schedule, and the one obtained by scheduling node *i* later. In the special case when the moment *t* is the latest possible moment, the node must be scheduled, so its priority becomes infinite:

$$f(i,t) = \begin{cases} \overline{E^i_t} - \overline{E^i_{t+1}} & \text{if } ALAP_i > t \\ \infty & \text{otherwise} \end{cases}$$

A negative priority means that it is better to schedule the node later, while a positive value means that it is better to schedule the node at that very moment. The priority function presented above considers only the energy aspect, and may fail to lead to feasible schedules, especially when the deadline is tight. To be able to find schedules even for tight deadlines, we used the fol-

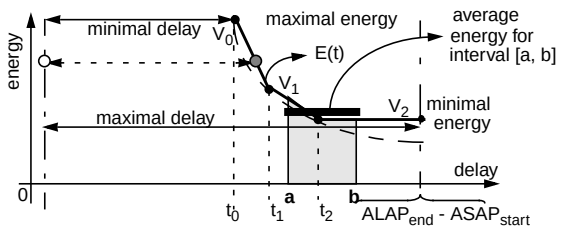


Fig. 4. Task energy relative to its execution time on a three voltage processor.

lowing priority function:

$$g(i,t) = f(i,t) + \alpha_i \cdot \frac{|f(i,t)|}{deadline - t - criticalpath(i)}$$

*Criticalpath(i)* is the delay of the longest path starting in node *i*. Each node has an associated coefficient $\alpha_i$, which controls the emphasis on lowering energy vs. generating a tight schedule. Having a different $\alpha$ for each node allows us to treat the nodes on the critical path in a different manner, focusing, for those nodes, more on fulfilling the deadline than on lowering the energy. With the priority given above, if all $\alpha_i$ are large enough, the priority function behaves as a classic, critical-path priority. Moreover, the set of smallest $\alpha_i$ for a given graph and certain deadline yields the lowest energy consumption for that graph and deadline. Details about tuning the values in the $\alpha_i$ set are given in the next section.

### C. Tuning The Priority Function

Depending on the values for the $\alpha$ coefficients, it can happen that no schedule is found. In that case, the $\alpha$'s for the nodes on the critical path are increased, thus emphasizing the timing aspect of the priority function. A new scheduling is attempted with the new $\alpha$ values. In the worst case all $\alpha$ reach their maximal value, *Max*$\alpha$, set by the designer, and the *g(t, i)* priority function becomes a classic critical-path priority function.

A pseudo-code description of the LEneS algorithm is given in Fig. 5. The algorithm consist of list-scheduling using our priority function, wrapped in a tuning algorithm for the $\alpha_i$ coefficients. The complexity analysis of the LEneS algorithm

```
procedure LEneS(ETG)
  set all {αi}= 0;
  while list-scheduling(ETG, g) fails do
    let {αk} be the coefficients of the nodes on the critical path
    if all {αi} >= Maxα then scheduling fails
    else if all {αk} = Maxα then increase all αi by 10%
    else increase only {αk} by 10%
end procedure LEneS
```

Fig. 5. LEneS: the low-energy directed scheduling.

shows that it has a computational complexity of $O(V \cdot M \cdot N^3 \cdot \log(Max\alpha))$, where *N* is the number of nodes in the ETG, *M* is the number of time steps in the tightest deadline, *V* is the highest number of supply voltages supported by a processor, and *Max*$\alpha$ is the maximal value allowed for the $\alpha$'s.

## VII. Experimental Results

The first experiment evaluates the LEneS algorithm from the scheduling speed point of view. The results are represented in Fig. 6. The points in the base plane of the 3D graph depict the system configuration: number of processors and the distribution of tasks on the processors. For each of the configurations we generated hundred random task-graphs and then used LEneS to schedule them, obtaining an average scheduling time. The average time needed to perform the scheduling is represented on the vertical axis. Using interpolation, we obtained the dotted curves on the surface. The curves mark different time levels, ranging from 1 second to 10 minutes. For example, for

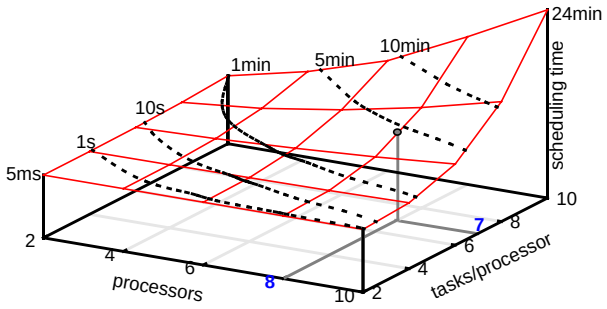


Fig. 6. The time required by LEneS to schedule different random task-graphs on various numbers of processors.

a task-graph of 56 tasks, evenly distributed on eight processors (seven tasks/processors), LEneS will require around 5 minutes to find the schedule. For scheduling the largest type of graphs (hundred nodes on ten processors), LEneS requires around 24 minutes. For this experiment we assumed that the processors can run at three different supply voltages (3.3V, 2.1V, and 0.9V). The reported times were obtained on a Sun Ultra10 workstation (440MHz UltraSparcIIi processor, 256MB RAM).

The long execution time for large designs makes our method suitable only for final scheduling, and not for fast evaluation inside a design-space exploration loop. Yet, this drawback can be overcome, if LEneS is combined with a simpler scheduling strategy or a fast estimator, as we point out in [10].

The next set of experiments inspects the energy saving capability of the LEneS algorithm compared to the classic list-scheduling with critical-path based priority function (ClassicLS). For several system configurations, similarly to the previous experiment, we scheduled the ETGs using both LEneS and ClassicLS. For the tightest schedule length, we compared the energies consumed by the two solutions, obtaining a surface similar to the one in Fig. 6. For clarity we depicted in Fig. 7 only the projection of the levels on the horizontal plane instead of the whole 3D graph. For this experiment, we assumed that we use processors with four supply voltages (3.3V, 2.5V, 1.7V, and 0.9V). Note that the saved energy can be as high as 28% when using LEneS as opposed to ClassicLS. For architectures with two voltage processors (3.3V and 0.9V), we obtained smaller energy savings. In the majority of the cases, the saved energy was four times smaller compared to the four supply voltage processors. This comes from the fact that for a two voltage processor, the associated energy-delay curve (Fig. 4) is a worse approximation of the ideal one, compared to the case of a four voltage processors.

The saved energy increases with the degree of parallelism (more processors or less tasks/processor). This comes from the fact that the percent of tasks on the critical-path decreases. In this case, there are potentially more tasks which can run slower, and thus save energy. The critical-path length is also influenced by the assignment of tasks to processors, not only by the dependencies in the task graph. A bad assignment can overload a processor unnecessary, increasing also the critical-path. In these cases, our LEneS method performs extremely well since it can take advantage of the idle processors. On the other hand,

if the processors are perfectly balanced, LEneS behaves as ClassicLS. A more detailed analysis of the influence of assignment on scheduling is given in [10].

In the experiments presented until now, we assumed that we always have to execute the task-graph as fast as possible. In reality, in most of the cases the deadlines are given as design requirements. Thus, often there is a time slack which can be used to further reduce the energy. The third experiment explores the behavior of LEneS in these cases. We considered three scheduling methods, based on the observation made in section IV: ClassicLS, ClassicLS with scaling (ScaledLS), and LEneS with scaling (ScaledLEneS). For various extensions of the tightest deadline, allowed by the time slack, we compared the energy saved by using ScaledLEneS over the other two approaches for a number of random task-graphs. In Fig. 8 we depicted the curves obtained by averaging the results for two sets of thirty random graphs (TG1 and TG2). Both of the two sets contain task-graphs of thirty nodes, but the degree of parallelism differs, representing two extremes. TG1 uses ten processors (high parallelism) while TG2 uses only three processors (low parallelism). The continuous curves show the energy saved by using ScaledLEneS over ScaledLS. The dotted curves show the energy saved by ScaledLEneS over ClassicLS. Note that ScaledLS preforms well, being able to save around 60% energy at 50% deadline extension compared to ClassicLS. Nevertheless, ScaledLEneS performs best, saving from 7% up to 28% energy, compared to ScaledLS.

The final experiment explores the energy saving possibility for a real-life application. The sub-system we are interested in is an optical flow detection (OFD) algorithm, which is part of a traffic monitoring system (see [14]). In the current implementation, the optical flow algorithm consists of 32 tasks, running on two ADSP-21061L digital signal processors. Limited by other tasks, OFD can process 78x120 pixels at a rate up to 12.5Hz. The estimated energy consumption for one iteration of the OFD implementation is 27.2µJ. Depending on car speed or monitoring altitude, such a high processing rate is a waste of resources. In many cases, a rate of 2Hz is sufficient, which means ca. six times deadline extension for one iteration of the OFD. Important energy savings would be obtained if the design were to use processors supporting multiple voltages. Moreover, using the approach we presented here, the schedule can be dynamically adapted at runtime to fit the desired deadline,

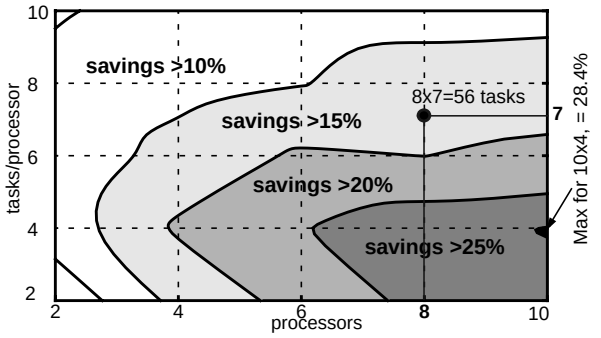


Fig. 7. The energy savings obtained when using LEneS over ClassicLS for different designs and configurations.

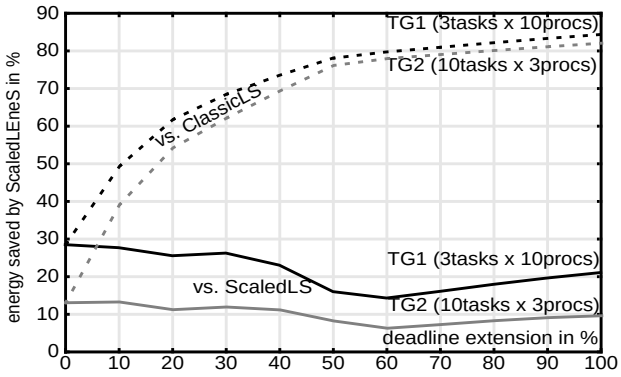


Fig. 8. The energy saved using LEneS with scaling compared to list scheduling, for different deadline extensions and system configurations.

depending on the required processing rate. Assuming we can run the DSPs at 3.3V, 2.5V, 1.7V, or 0.9V, we applied low-energy scheduling method for the OFD. The results show as much as 83% energy saving possibility for a sampling rate of 2Hz. For that rate, all the tasks are using the two lowest possible voltages. Overall, the processors have to run 42.48% of the time at 1.7V, and the rest of the time at 0.9V. For this particular example, the difference between the energy obtained through ScaledLEneS and ScaledLS was less than 4%. This difference is rather small because of the reduced number of tasks executing off the critical path, and which anyway have little freedom to scale. Note also that as the deadline extension grows, the tasks use lower voltages. For lower supply voltages, the energy-delay dependency (Fig. 4) has a smaller slope, so longer delays yield gradually smaller energy savings. At limit, all tasks execute at the lowest voltage, and any further deadline extension will not bring any energy saving.

## VIII. Conclusion

Since processors with dynamic supply voltage start to be available on the market, there is a need for new methods targeting reduced energy consumption, while taking advantage of the features of these new types of processors. In this context, we present a number of scheduling techniques targeting energy consumption reduction. We introduce our new scheduling approach, LEneS, that handles designs with dependent tasks mapped onto dynamic supply voltage processors. LEneS uses list-scheduling and a special priority function to derive static schedules with low energy consumption. Using a method for tuning its priority function, our algorithm is able to find schedules that are more energy efficient than the other mentioned approaches. The experiments presented in this paper show that even for the tightest possible deadline, up to 28% energy savings can be obtained without any performance loss, using our scheduling approach. For loose deadlines we proposed a scaling method which yields important energy savings and can be applied in principle to any scheduling algorithm. In particular, using this technique, we can obtain 77% energy savings over a critical-path priority based list-scheduling, for a 50% deadline extension.

## APPENDIX

In this part we prove, using the formulae introduced in section III, that scaling the tightest schedule to the desired deadline is the optimal strategy from the energy point of view. We start from the task *i* energy expression for two different supply voltages $V_0$, the reference voltage, and $V$, the voltage after scaling:

$$E_{0i} = N_i \cdot K_a \cdot {V_0}^2,\ E_i = N_i \cdot K_a \cdot V^2 \quad (1)$$

For a sufficiently small threshold voltage, the clock frequencies for the two voltages are (section III):

$$f_0 = K_b \cdot V_0,\ f_i = K_b \cdot V \quad (2)$$

From (1), (2): $E_i = E_{0i} \cdot (f_i / f_0)^2$ (3)

From section III: $E_{0i} = (N_i \cdot \pi_0) / f_0$ (4)

From (3), (4): $E_i = N_i \cdot \pi_0 \cdot f_i^2 / f_0^3$ (5)

If $\delta_i$ is the task execution delay: $f_0 = \frac{N_i}{\delta_{0i}},\ f_i = \frac{N_i}{\delta_i}$ (6)

From (5), (6): $E_i = \pi_0 \cdot \delta_{0i}^3 / \delta_i^2$ (7)

The total energy of the task graph is:

$E = \sum E_i = \pi_0 \cdot \sum \delta_{0i}^3 / \delta_i^2$ where we know that the tasks execute during the whole time until the deadline $\sum \delta_i = d$, and the tightest possible deadline is $\sum \delta_{0i} = d_0$.

The lower bound for the total energy is $\pi_0 \cdot d_0^3 / d^2$, provable by mathematical induction. This lower bound can be obtained only when $\delta_i = \delta_{0i} \cdot d / d_0$. Thus by scaling the tightest schedule to fit the new deadline. The supply voltage associated with the new schedule can be easily computed.

## ACKNOWLEDGMENT

Special thanks to Petru Eles and Jonas Hallberg for their insightful comments that helped us to improve this paper. The work presented here was partially sponsored by ARTES: A network for Real-Time research and graduate Education in Sweden, http://www.artes.uu.se/, and WITAS [14].

## REFERENCES

[1] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in CMOS circuits," *Proc. of the IEEE*, Vol. 83, No. 4, pp. 498-523, 1995.

[2] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable pomputation," *IEEE Trans. on VLSI Systems*, Vol.4 No.1, pp. 42-51, 1996.

[3] C. Hwang and A. C. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design 1997*, pp. 28-32.

[4] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *Proc. of the 36th Symposium on Foundations of Computer Science*, pp. 374-382, 1995.

[5] I. Hong, Gang Qu, M. Potkonjak, and M.B. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processors," *Proc. of the 19th IEEE Real-Time Systems Symposium 1998*, pp.178-187.

[6] I. Hong, D. Kirovski, Gang Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable voltage core-based systems," *Proc. of the 35th Design Automation Conference 1998*, pp. 176-181.

[7] T. Ishihara,H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, 1998, pp 197-202.

[8] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: hardware-software co-synthesis of embedded systems", *Proc. of the 34th DAC 1997*, pp. 703-708.

[9] F. Gruian and K. Kuchcinski, "Low-energy directed architecture selection and task scheduling for system-level design," *Proceedings of the 25th Euromicro Conference 1999*, pp. 296-302.

[10] F. Gruian, "System-level design methods for low-energy architectures containing variable voltage processors," in press.

[11] J.-M. Chang and M. Pedram, "Energy minimization using multiple supply voltages," *IEEE Trans. on VLSI Systems*, Vol. 5, No. 4, 1997.

[12] K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, and T. Kuroda, "A 300MIPS/W RISC core processor with variable supply-voltage scheme in variable threshold-voltage CMOS," *Proc. of the IEEE Custom Integrated Circuits Conference 1997*, pp. 587-590.

[13] "SH-1:SH7032/SH7034 Product Brief," *HITACHI Semiconductor Inc.*

[14] WITAS: The Wallenberg laboratory for research on information technology and autonomous systems, http://www.ida.liu.se/ext/witas/