Multiprocessor Scheduling to Reduce Leakage Power

Pepijn de Langen Ben Juurlink Stamatis Vassiliadis
Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
Phone: +31-15-278-3644
E-Mail:{pepijn|benj|stamatis}@ce.et.tudelft.nl

Abstract— **In contemporary and future embedded as well as high-performance microprocessors, power consumption is one of the most important design considerations. Because in current technologies the dynamic power consumption dominates the static power consumption, voltage scaling is an effective technique to reduce the power consumption. The most common way to reduce the power consumption of multi-processor systems, is to schedule a program to run on as many processors as possible and apply voltage scaling afterwards. As technology scales to increasingly smaller feature sizes, however, the static power consumption is expected to grow exponentially. In this paper, we first show for which combinations of leakage current, supply voltage, and clock frequency the static power consumption dominates the dynamic power dissipation. Based on these results, it is at a certain point no longer advantageous to use as many processors as possible. We then present a heuristic to schedule task graphs on a number of processors that is sufficient to meet the deadline, but at the same time minimizes the power consumption. Experimental results have been obtained using our in-house power analysis tool with task graphs from real applications as well as randomly generated ones. The results show that our scheduling algorithm reduces the total energy consumption by up to 65%, compared to the strategy that schedules the tasks on the maximum number of processors and then exploits the remaining slack to lower the supply voltage.**

Keywords: multiprocessor systems, scheduling, leakage power, voltage scaling

I. Introduction

Currently, power consumption is one of the most important issues in the design of microprocessors. Not only does this apply to high-performance embedded processors in battery powered devices, also in desktop machines and high-performance dedicated systems power consumption is a fundamental problem that limits clock frequencies and scaling of distributed systems. Through the advent of (single chip) multiprocessors for the embedded market, power consumption is becoming increasingly important for multiprocessor systems as well. Power consumption can generally be classified in dynamic and static power consumption. The first relates to the power that is dissipated due to switching activity, while the second one is due to leakage currents.

Because in current technologies the dynamic power consumption dominates the static power consumption, and because the dynamic power dissipation grows quadratically with the supply voltage, voltage scaling is an effective technique to reduce the power consumption. Consequently, when scheduling tasks on a multiprocessor system, it is advantageous to employ as many processors as possible so that the remaining slack can be exploited to lower the supply voltage. While in the past static power consumption could be ignored, it should not be neglected in the near future. With decreasing feature sizes, leakage current is expected to increase exponentially [7] in the next decade. Because of this, it will not always be advantageous to use as many processors as possible. In this work, we present a scheduling algorithm that is targeted at a near future technology, where leakage current is responsible for at least 50% of the total amount of dissipated power. The algorithm we present schedules task graphs on a number of processors that is sufficient to meet the deadline, while the total power consumption is minimized.

This paper is organized as follows: Section II contains a brief overview of related work. In Section III, we will describe the conditions under which voltage scaling can be applied to reduce energy consumption. Section IV describes our scheduling and voltage selection algorithm. Experimental results are given in Section V. Section VI finishes with the conclusions and directions for future research.

II. Related Work

Reducing power consumption has been an important research topic in the past years, both in em-

bedded systems and in high-performance related research. One of the most promising techniques that have been proposed in this area is dynamic voltage scaling (DVS), where both the clock frequency and the supply voltage are scaled down.

The combination of dynamic voltage scaling and multi-processor scheduling has been investigated by a significant number of researchers in the past years. Several authors (e.g. [4], [13]) used existing scheduling techniques, such as list scheduling with earliest deadline first, to finish the tasks as early as possible and used the remaining slack to lower the supply voltage. We will refer to this strategy as *Schedule and Stretch* (S&S). In other work [12], the scheduling is done in a way to optimize the possibilities for selecting different voltages. Varatkar et al. [10] included communication in their power estimations and tried to execute part of the code on a lower supply voltage while minimizing communication.

Jejurikar et al. [5] included leakage current in their energy estimations and proposed to maximize slack time to allow processors to shut down temporarily. This was combined with DVS and used for real-time scheduling.

Some researchers have proposed to also adjust the threshold voltage when scaling the supply voltage [3], [9]. Others have extended this to scheduling for real-time multi-processor systems [1], [11].

Other means of reducing a processors power consumption are mostly targeted at caches, since these structures require a significant portion of the area on a chip, and are responsible for a large part of the total amount of dissipated power. Therefore, an effective way to reduce the amount power dissipated in these parts is to shut down parts of the cache [2], [7].

Our work differs in the following ways. First of all, we use experimental models that are projected in the near future, where leakage current is of much more importance than it is in current technologies. Second, we employ frequency and voltage scaling concurrently to all processors. In other words, only one voltage/frequency pair is chosen for all processors for the duration of the whole schedule. Third, we do a full search to find the optimal number of processors. Fourth, most research mentioned above target scheduling of real-time recurring tasks, while we target scheduling of a task graph.

III. CPU Energy Consumption

In this section, we will first derive how static and dynamic power dissipation relate to leakage-current, supply voltage, and clock frequency. From this, we then derive the extend to which voltage scaling is effective for a certain processor. In the second part, we will show how these results can be used for scheduling on multi-processor systems.

A. Voltage Scaling Requirements

An approximation for the power consumption in a CMOS gate is:

$$P = C_L V^2 f + I_q V \quad , \tag{1}$$

where C_L is the load capacitance, V is the supply voltage, I_q is the leakage current, and f is the operating frequency. The first term in this equation corresponds to the amount of dynamically dissipated power, caused by switching circuitry. The second part models the amount of statically dissipated power, generated by leakage current.

First, we will look at what the requirements are for voltage scaling to be beneficial for the total energy consumption. For this purpose, we will first derive an expression for the normalized amount of power dissipation.

First, we split Equation (1) into a part for dynamic dissipation (D) and one for static dissipation (S):

$$P = D + S \tag{2}$$

$$D = C_L V^2 f \tag{3}$$

$$S = I_q V \quad . \tag{4}$$

To normalize these expressions, we define that at maximum frequency f_{max} and corresponding supply voltage V_{max}, a processor will dissipate an amount of power $P_{max} = D_{max} + S_{max}$. With $\mathcal{P}$, $\mathcal{D}$, and $\mathcal{S}$ denoting the normalized total, dynamic, and static power dissipation, we then write:

$$\mathcal{P} = \frac{P}{P_{max}} = \mathcal{D} + \mathcal{S} = \frac{D}{P_{max}} + \frac{S}{P_{max}} \quad . \tag{5}$$

We then define δ and σ as:

$$\delta = \frac{D_{max}}{D_{max} + S_{max}}, \quad \sigma = \frac{S_{max}}{D_{max} + S_{max}} \quad . \tag{6}$$

In other words, δ and σ denote how much of the total power dissipation at maximum frequency is caused by switching activity and how much by leakage current.

Let $\mathcal{V} = V/V_{max}$ be the normalized voltage and $\mathcal{F} = f/f_{max}$ the normalized frequency. The expressions for normalized dynamic and static dissipation can then be rewritten as:

$$\mathcal{D} = \delta \frac{D}{D_{max}} = \delta \frac{C_L V^2 f}{C_L V_{max}^2 f_{max}} = \delta \mathcal{V}^2 \mathcal{F} \quad , \tag{7}$$

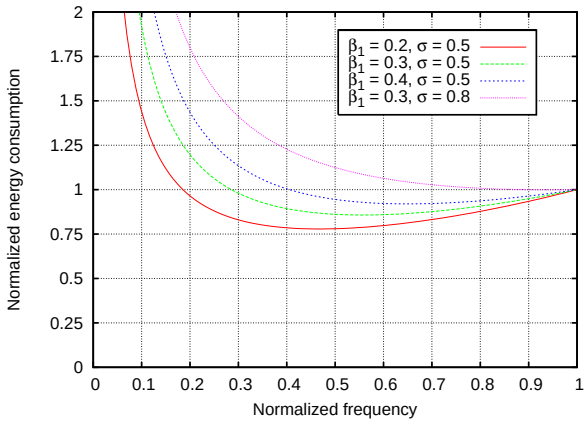

Fig. 1. Normalized Energy Consumption

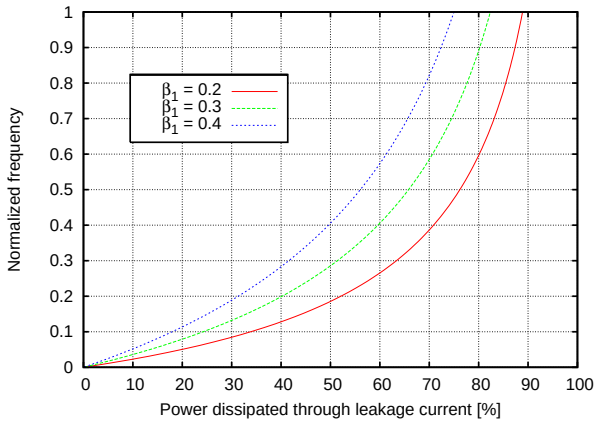

Fig. 2. Minimum Energy-Conserving Frequencies

and

$$\mathcal{S} = \sigma \frac{S}{S_{max}} = \sigma \frac{I_q V}{I_q V_{max}} = \sigma \mathcal{V} \quad . \tag{8}$$

Combining Equation (5) with Equations (7) and (8) then results in:

$$\mathcal{P} = \delta \mathcal{V}^2 \mathcal{F} + \sigma \mathcal{V} \quad . \tag{9}$$

Using a normalized expression for time $\mathcal{T} = 1/\mathcal{F}$, the expression for the normalized energy consumption $\mathcal{E}$ then becomes:

$$\mathcal{E}(\mathcal{F}, \mathcal{V}) = \mathcal{P}(\mathcal{F}, \mathcal{V})\mathcal{T} = \delta \mathcal{V}^2 + \sigma \mathcal{V}/\mathcal{F} \quad . \tag{10}$$

From (10), it is clear that voltage scaling is only beneficial if for a certain $\mathcal{F} < 1$ there exists a $\mathcal{V} < 1$, so that:

$$\mathcal{E}(\mathcal{F}, \mathcal{V}) < 1 \quad . \tag{11}$$

The supply voltage of a processor must be high enough to guarantee that the logic levels are always safely reached before the end of a clock cycle. This implies that the required supply voltage actually depends on the operating frequency. From [8], we take the following expression approximating the relation between normalized frequency and voltage:

$$\mathcal{V} = \beta_1 + \beta_2 \mathcal{F} \quad , \tag{12}$$

where $\beta_1 = V_{th}/V_{max}$ and $\beta_2 = 1 - \beta_1$, with V_{th} being the threshold voltage and V_{max} the voltage at maximum frequency. Again, $\mathcal{F}$ represents the normalized frequency.

Combining this with Equation (10) leads to:

$$\begin{aligned} \mathcal{E}(\mathcal{F}) =& \delta(\beta_2^2 \mathcal{F}^2 + 2\beta_1\beta_2\mathcal{F} + \beta_1^2) + \\ & \sigma(\beta_1/\mathcal{F} + \beta_2) \quad . \end{aligned} \tag{13}$$

In Figure 1, the normalized energy consumption is plotted against the frequency for a number of different configurations. The first three curves depict systems where, at maximum frequency, the leakage current is responsible for 50% of the total energy consumption. The fourth curve depicts a system where the amount of energy consumed by leakage currents is 80% of the total. In this case, it is hardly possible to effectively employ voltage scaling, unless the threshold voltage is decreased. From Figure 1, one can see that a higher threshold voltage diminishes the possibility to effectively employ voltage scaling.

In order to show the limits of voltage scaling, we have included Figure 2. In this figure, we have plotted σ versus $\mathcal{F}$, so that $\mathcal{E}$ in Equation (13) equals 1. We have included curves for three different relative threshold voltages ($\beta_1 = V_{th}/V_{max}$). Furthermore, we have swapped the axes to show the minimum frequencies to where voltage scaling can be effective for a certain a priori chosen percentage static dissipation (σ). Areas to the right (more leakage current) or below (lower frequency) the lines in this graph depict situations where voltage scaling actually increases the energy consumption.

It might not always be possible to scale down enough, based on the threshold voltage and the amount of leakage current. Under those circumstances, the most energy efficient strategy would be to finish the task at a higher frequency, and turn to a low-power (sleep) mode for the remaining time. Such power saving modes, in fact, can incur significant penalties when the involved logic has to be turned on again. This actually limits the possibility of effectively using sleep modes.

B. Parallelization and Voltage Scaling

In the previous section, we have determined the circumstances under which it is useful to employ voltage scaling in a single processor. In this section, we will assume that a task graph has a tight deadline, so that several processors must be used in order to meet this deadline. For a multiprocessor system, the requirements for lowering energy consumption by voltage scaling are equivalent as to the case with only one processor. For technologies with very low leakage current, where voltage scaling always decreases the energy consumption, the lowest-energy solution is to run the tasks on as many processors as possible, with the lowest possible frequency. On the other hand, when energy consumption cannot effectively be decreased by voltage scaling, the lowest-energy solution is to run the tasks on as few processors as possible. In this work, we assume a technology with relatively high leakage current where the possibility to reduce energy consumption by voltage scaling is limited.

When running programs in parallel, it is by definition impossible to achieve a super-linear speedup. In fact, linear speedup is only theoretically possible and will never be achieved for real programs. For example, when increasing the number of processors by a factor of two, the number of required cycles will decrease by a factor less than two. Because of this, it is only beneficial to increase the number of processors if there is enough parallelism exploitable.

The amount of available slack is often not sufficient to make it profitable to put a processor into sleep mode. Especially for multiprocessor systems, the total amount of slack is often fragmented across different processors and over time. Under these circumstances, it is beneficial to use voltage scaling to exploit the available slack, even though the power dissipation through leakage current can be very high. Although in some cases it might turn out that sleep modes could be more beneficial than frequency scaling, we will not consider this, since it is outside of the scope of this work.

IV. Scheduling for Low Power

As described in the previous section, it is not always beneficial to the total amount of power consumption to employ as much parallelism as possible.

In our approach, we take the following assumptions: We assume that all used processors are in active mode all the time. In other words, it is impossible to turn used processors into sleep mode. Second, we assume that all processors in one system run at the same frequency. Furthermore, we have not included any communication delays between the nodes.

For the case where voltage scaling would increase energy consumption, finding the optimum number of processors is just the same as finding the minimum number of processors that can finish the tasks before the given deadline. For processors with a less disastrous leakage current, this number depends on the amount of parallelism that can be exploited. As explained before, in this work we will use a processor for which it is still possible to successfully employ voltage scaling, however to a very limited extend. This depends for a large part on the possibility to exploit available parallelism.

The basis of our approach is a list-scheduling algorithm that employs an *Earliest-Deadline-First* (EDF) priority function. After scheduling, the remaining cycles before the deadline (slack) is used to lower the clock frequency and supply voltage. This implies that any resulting schedule will always finish exactly at the deadline. The normalized clock frequency for a schedule can then be found by dividing the number of cycles by the deadline. Since the different schedules produced by this algorithm all finish at the same time, we can compare different solutions by power consumption instead of energy consumption.

To determine the optimum number of processors, we first define the theoretical minimum (N_{min}) and the theoretical maximum number of processors (N_{max}) as:

$$N_{min} = \lceil \frac{\Sigma_{work}}{\text{deadline}} \rceil \quad , \tag{14}$$

$$N_{max} = \text{number\_of\_tasks} - 1 \quad , \tag{15}$$

where Σ_{work} is the sum of the weights of all nodes. Then, we determine the minimum number of processors that is required to finish the tasks before the deadline, using a binary search on an interval from the theoretical minimum to the theoretical maximum. Starting with this minimum, we then generate schedules for increasing numbers of processors. This loop is exited whenever increasing the number of processors no longer decreases the required number of cycles. At this point, the graph is scheduled at exactly the length of the critical path. For each generated schedule, the total power consumption is recorded. The algorithm finishes by selecting the configuration that requires the least amount of power.

The reason for doing a full search on the number of processors comes forth from the fact that minima in

name	nodes	edges	CP	Σwork
sparse	96	128	122	1920
robot	88	130	545	2459
fpppp	334	1196	1062	7113
proto001	273	1688	167	4711
proto003	164	646	556	1599
proto279	1342	16762	735	13302

TABLE I
BENCHMARKS

these results are not necessarily global minima. This will be shown in more detail in Section V.

In order to determine the power consumption of a multiprocessor system, we first derive an expression similar to Equation (9) for a multiprocessor system consisting of N processors:

$$\mathcal{P}_{multi} = N(\delta\mathcal{V}^2\mathcal{F} + \sigma\mathcal{V}) \quad . \tag{16}$$

Combining this with Equation (12) then results in:

$$\mathcal{P}_{multi} = N(\delta\beta_2^2\mathcal{F}^3 + 2\delta\beta_1\beta_2\mathcal{F}^2 + \delta\beta_1^2\mathcal{F} + \sigma\beta_2\mathcal{F} + \sigma\beta_1) \quad . \tag{17}$$

V. Experimental Results

In this section, we present the results of our scheduling approach.

In these experiments, we will assume a technology where leakage current contributes to the overall power consumption to a much larger extend than it does today. In fact, we state that half of the power consumption at maximum frequency is due to this leakage current ($\delta = 0.5, \sigma = 0.5$). Furthermore, we assume that β_1 and β_2 in Equation (12) are 0.3 and 0.7 respectively. Filling in these values in (17), gives the equation for the total amount of power:

$$\mathcal{P}_{multi} = N(0.245\mathcal{F}^3 + 0.21\mathcal{F}^2 + 0.395\mathcal{F} + 0.15) \quad . \tag{18}$$

Note that, although we have assumed a relatively high leakage current in this processor, according to Equation (13) and Figure 2, it is still theoretically possible to reduce energy consumption by scaling the frequency down by even more than a factor of 2.

Table I lists the benchmarks that were used, along with the number of nodes and edges, as well as the length of the critical path (CP), and the total weight of all the nodes (Σwork). The first three benchmarks have been derived from real applications, while the other three have been randomly generated. All of these benchmarks were taken from the *Standard Task Graph Set* [6]. Since there is no real deadline for any of these experiments, we have used deadlines of 1.5, 2, 4, and 8 times the length of the critical path (CP).

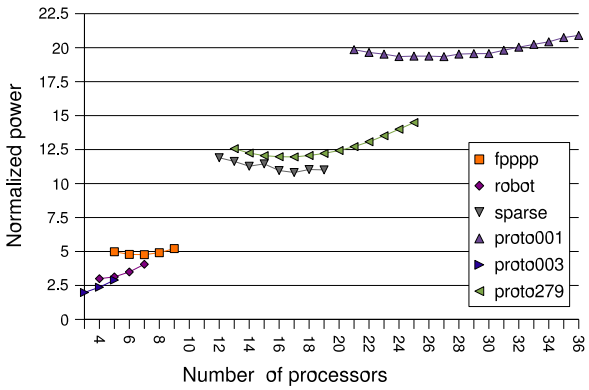

Fig. 3. Normalized energy consumption for different benchmarks with the deadline at 1.5 times the critical path length.

Figure 3 depicts the relative power consumption for different task graphs, scheduled at all possible numbers of processors for a deadline that is 1.5 times the weight of its critical path. From this figure, we can see that there are local minima that are not globally optimal, for example with the *sparse* benchmark on 15 processors. Therefore, a full search must be performed on the number processors, in order to find the optimum for a certain graph and deadline.

In Tables II to V, the results of our experiments are depicted. We have used two different algorithms in our experiments. The first one is our algorithm, as described in Section IV. We will refer to this algorithm as *Minimum Power Scheduling* (MPS). The second algorithm we use is often used in DVS-related research, for example in [4]. In this algorithm, a task graph is first scheduled using *list scheduling*, using *earliest-deadline-first* as the priority assignment. The remaining slack is then used to stretch the whole schedule. We will refer to this algorithm as *Schedule-and-Strech* (S&S). The main difference between the two algorithms is that with *Schedule-and-Stretch*, the graph is scheduled on the maximum number of processors that is useful to decrease the length of the schedule. In our algorithm, on the other hand, we use the number of processors that minimizes the power consumption. For each benchmark, we have listed the number of processors (N), the normalized frequency ($\mathcal{F}$), and the total normalized power consumption ($\mathcal{P}_{multi}$). Note

benchmark	MPS N	MPS $\mathcal{F}$	MPS $\mathcal{P}$	S&S N	S&S $\mathcal{F}$	S&S $\mathcal{P}$
fpppp	7	0.76	4.76	9	0.67	5.21
robot	4	0.82	3.00	7	0.67	4.06
sparse	17	0.72	10.81	19	0.67	11.01
proto001	27	0.79	19.34	36	0.67	20.90
proto003	3	0.74	1.99	5	0.67	2.90
proto279	17	0.78	11.96	25	0.67	14.49

TABLE II
RESULTS FOR DEADLINE AT 1.5 TIMES THE LENGTH OF THE CRITICAL PATH

benchmark	MPS N	MPS $\mathcal{F}$	MPS $\mathcal{P}$	S&S N	S&S $\mathcal{F}$	S&S $\mathcal{P}$
fpppp	5	0.75	3.33	9	0.50	3.88
robot	3	0.77	2.07	7	0.50	3.01
sparse	14	0.65	7.84	19	0.50	8.18
proto001	21	0.72	13.36	36	0.50	15.50
proto003	2	0.75	1.35	5	0.50	2.15
proto279	14	0.69	8.38	25	0.50	10.77

TABLE III
RESULTS FOR DEADLINE AT 2 TIMES THE LENGTH OF THE CRITICAL PATH

that, in case of the *Schedule-and-Stretch*, the clock frequency can be derived directly from the length of the critical path (CP) and the chosen deadline:

$$f_{s\&s} = \text{CPW}/\text{deadline} \quad . \tag{19}$$

In Figure 4, the improvements made by our method are depicted. From this figure it can be seen that a significant amount of power can be saved. For tight deadlines, the savings range from 2% to 31%. For less strict deadlines, the savings increase to more than 65%. Because the processors cannot always be used to their full extend, the low-power solution in most experiments is to use typically less than maximum number of processors. Both the optimal number of processors and the resulting power consumption are largely depending on the chosen deadline.

VI. Conclusions & Future Work

As feature sizes keep decreasing, the contribution of leakage current to the total energy consumption is expected to increases significantly. In this work, we have shown that soon the point will be reached where maximizing the amount of parallelism is not always beneficial from an energy saving perspective.

benchmark	MPS N	MPS $\mathcal{F}$	MPS $\mathcal{P}$	S&S N	S&S $\mathcal{F}$	S&S $\mathcal{P}$
fpppp	3	0.58	1.48	9	0.25	2.39
robot	2	0.57	0.98	7	0.25	1.86
sparse	6	0.67	3.51	19	0.25	5.05
proto001	12	0.60	6.24	36	0.25	9.57
proto003	1	0.72	0.63	5	0.25	1.33
proto279	8	0.58	4.01	25	0.25	6.64

TABLE IV
RESULTS FOR DEADLINE AT 4 TIMES THE LENGTH OF THE CRITICAL PATH

benchmark	MPS N	MPS $\mathcal{F}$	MPS $\mathcal{P}$	S&S N	S&S $\mathcal{F}$	S&S $\mathcal{P}$
fpppp	2	0.42	0.75	9	0.12	1.83
robot	1	0.56	0.48	7	0.12	1.42
sparse	3	0.66	1.71	19	0.12	3.86
proto001	6	0.59	3.04	36	0.12	7.31
proto003	1	0.36	0.33	5	0.12	1.02
proto279	4	0.57	1.97	25	0.12	5.08

TABLE V
RESULTS FOR DEADLINE AT 8 TIMES THE LENGTH OF THE CRITICAL PATH

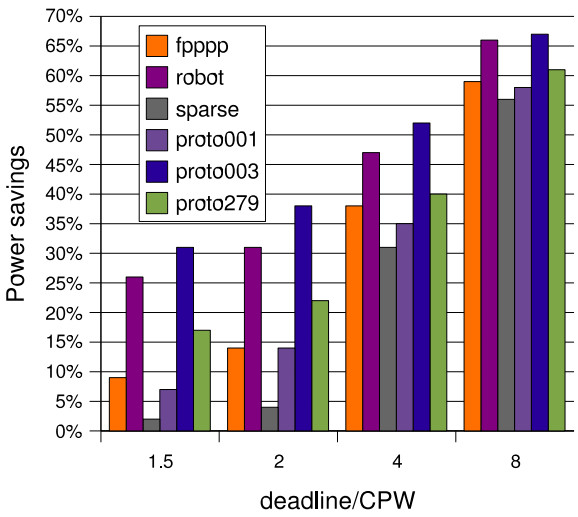

Fig. 4. Improvements of our proposed method compared to S&S.

Depending on the amount of leakage current and an applications parallelism, a combination of parallelism and clock-speed can be determined that minimizes energy consumption. In our experiments, we have shown that our solution can save over 30% of the amount of dissipated power for tight deadlines and over 65% for loose deadlines.

In the end, the relative amount of energy dissipated by leakage currents is expected to become much larger than the amount of energy dissipated through switching activity. At that point, the lowest power solution will be to perform as much as possible sequentially on one processor and to only use parallelism if the required performance demands to do so.

We have assumed that all processors in one system operate at the same frequency. By having some processors slow down more than others, it would be possible to make a more balanced schedule and in that way save more power. This is a goal for future research. Also the possibility to let processors to go into sleep mode is an interesting option.

Furthermore, it would be interesting to see how delays and additional power consumption, needed for communication between the processors, would influence these results.

References

[1] Alexandru Andrei, Marcus Schmitz, Petru Eles, Zebo Peng, and Bashir M. Al-Hashimi. Overhead-Conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time-Constrained Systems. In *Proc. Conf. on Design, Automation and Test in Europe*, page 10518, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Krisztin Flautner, Nam Sung Kim, Steve Martin, David Blaauw, and Trevor Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *Proc. Int. Symp. on Computer Architecture*, pages 148–157, Washington, DC, USA, 2002. IEEE Computer Society.

[3] R. Gonzalez, B. Gordon, and M. Horowitz. Supply and Threshold Voltage Scaling for Low Power CMOS. *IEEE Journal of Solid-State Circuits*, 32(8), 1997.

[4] F. Gruian and K. Kuchcinski. LEneS: Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processors. In *Proc. Conf. on Asia South Pacific Design Automation*, pages 449–455, 2001.

[5] Ravindra Jejurikar, Cristiano Pereira, and Rajesh Gupta. Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems. In *Proc. Conf. on Design Automation*, pages 275–280, New York, NY, USA, 2004. ACM Press.

[6] Hironori Kasahara, Takao Tobita, Takanari Matsuzawa, and Shinya Sakaida. Standard Task Graph Set. `http://www.kasahara.elec.waseda.ac.jp/schedule/`.

[7] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proc. Int. Symp. on Computer Architecture*, pages 240–251, 2001.

[8] Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge, Krisztián Flautner, Jie S. Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage Current: Moore's Law Meets Static Power. *IEEE Computer*, 36(12):68–75, 2003.

[9] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads. In *Proc. Int. Conf. on Computer-Aided Design*, November 2002.

[10] Girish Varatkar and Radu Marculescu. Communication-Aware Task Scheduling and Voltage Selection for Total Systems Energy Minimization. In *Proc. Int. Conf. on Computer-Aided Design*, page 510, Washington, DC, USA, 2003. IEEE Computer Society.

[11] Le Yan, Jiong Luo, and Niraj K. Jha. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-time Embedded Systems. In *Proc. Int. Conf. on Computer-Aided Design*, page 30, Washington, DC, USA, 2003. IEEE Computer Society.

[12] Yumin Zhang, Xiaobo Sharon Hu, and Danny Z. Chen. Task Scheduling and Voltage Selection for Energy Minimization. In *Proc. Conf. on Design Automation*, pages 183–188, New York, NY, USA, 2002. ACM Press.

[13] D. Zhu, R. Melhem, and B.R. Childers. Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(7):686–700, 2003.