Leakage-Aware Multiprocessor Scheduling for Low Power

Pepijn de Langen and Ben Juurlink

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 2783644, email: pepijn@ce.et.tudelft.nl

Abstract

It is expected that (single chip) multiprocessors will increasingly be deployed to realize high-performance embedded systems. Because in current technologies the dynamic power consumption dominates the static power dissipation, an effective technique to reduce energy consumption is to employ as many processors as possible in order to finish the tasks as early as possible, and to use the remaining time before the deadline (the slack) to apply voltage scaling. We refer to this heuristic as Schedule and Stretch (S&S). However, since the static power consumption is expected to become more significant, this approach will no longer be efficient when leakage current is taken into account. 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. These results imply that, at a certain point, it is no longer advantageous from an energy perspective to employ as many processors as possible. Thereafter, a heuristic is presented to schedule the tasks on a number of processors that minimizes the total energy consumption. Experimental results obtained using a public task graph benchmark set show that our leakage-aware scheduling algorithm reduces the total energy consumption by up to 24% for tight deadlines ($1.5x$ *the critical path length) and by up to 67% for loose deadlines (*$8x$ *the critical path length) compared to S&S.*

1 Introduction

In contemporary and future embedded as well as high-performance microprocessors, power consumption is one of the most important design considerations. Not only does this apply to processors embedded in battery powered devices, but also in desktop machines and high-performance dedicated systems power consumption is a fundamental problem that limits clock frequencies. Through the advent of (single chip) multiprocessors for the embedded market, such as the IBM/Sony/Toshiba Cell architecture [5] and Philips Wasabi [12], 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 in order to maximize the remaining time before the deadline. This slack can then be exploited to lower the clock frequency and supply voltage. However, as technology scales to increasingly smaller feature sizes, static power dissipation due to leakage current is expected to grow exponentially in the near future [9]. In this case, using as many processors as possible combined with voltage scaling will no longer provide an efficient solution. In other words, while in the past static power consumption could be ignored, it should not be neglected in the future.

In this paper, we present a scheduling algorithm that is targeted at a near future technology, where leakage current is responsible for a significant part of the total power dissipation. 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 2 contains an

This research was supported in part by the Netherlands Organisation for Scientific Research (NWO).

1-4244-0054-6/06/$20.00

overview of related work. In Section 3, we describe the conditions under which voltage scaling can be applied to reduce energy consumption. Section 4 describes our scheduling and voltage selection algorithm. Experimental results are provided in Section 5. In Section 6 conclusions are drawn and directions for future research are given.

2 Related Work

Reducing power consumption has been an important research topic in the past years, both in embedded 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.

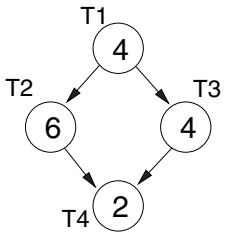
The combination of dynamic voltage scaling and multiprocessor scheduling has been investigated by a significant number of researchers in the past years. Jha [7] provides a detailed overview of work in this area.

A technique proposed by several authors [4, 16] is to use existing scheduling techniques, such as list scheduling with earliest deadline first (EDF), to finish the tasks as early as possible and use the remaining time to lower the supply voltage. However, these authors did not include leakage current in their power estimations. Several authors have proposed this technique using different names and, therefore, we refer to it as *Schedule and Stretch* (S&S). Figure 1 illustrates the concept behind S&S. First, the task graph in Figure 1(a) is scheduled in a way that minimizes the length of the schedule, as depicted in Figure 1(b). From this figure, it can be seen that after the scheduling process, there are certain periods in which a processor is idle. This idle time is often referred to as *slack*. In the S&S algorithm, the power consumption is decreased by using the slack that remains at the end of the schedule to lower the clock frequency and supply voltage of all processors, as depicted in Figure 1(c).

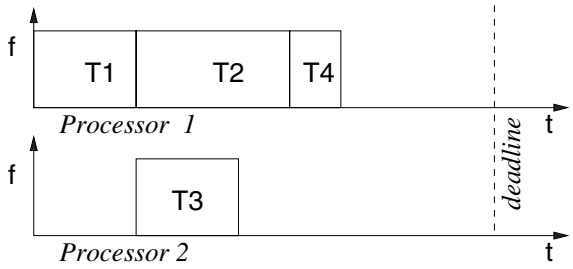
Jejurikar et al. [6] included leakage current in their energy estimations and proposed to maximize slack time to allow processors to shut down temporarily. This was combined with DVS and used for real-time scheduling. These authors, however, assumed tasks that are independent.

In other work [15], the scheduling is done in a way to optimize the possibilities for selecting different voltages. Varatkar et al. [13] included communication in their power estimations and tried to execute part of the code on a lower supply voltage while minimizing communication.

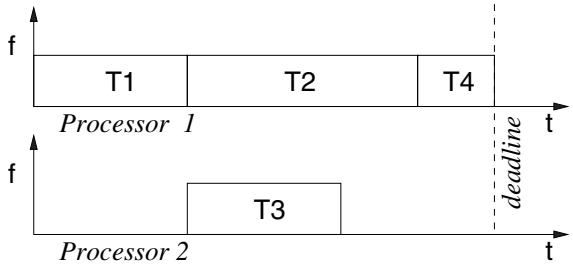
Some researchers have proposed to also adjust the threshold voltage when scaling the supply voltage [3, 11]. Others have extended this to scheduling for real-time multiprocessor systems [1, 14]. Both these authors as the ones mentioned above, however, did not optimize on the the number of employed processors.



(a) The task graph



(b) Schedule produced by EDF



(c) Stretched schedule

Figure 1. Illustration of the Schedule and Stretch algorithm.

Other techniques to reduce the energy consumption of a processor 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 power consumption is to shut down parts of the cache [2, 9].

3 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 can be used to decrease

the energy consumption for a certain processor. In the second part, we will show how these results can be used for scheduling on multi-processor systems.

3.1 Voltage Scaling Requirements

The power consumption in a CMOS gate can be approximated by:

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

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

We will start with looking 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.

To normalize Expression (1), we define that at maximum frequency f_{max} and corresponding supply voltage V_{max}, a processor will dissipate an amount of power $P_{max} = D_{max} + S_{max}$. The normalized total, dynamic, and static power dissipation ($\mathcal{P}$, $\mathcal{D}$, and $\mathcal{S}$) can then be written as:

$$\mathcal{P} = \frac{P}{P_{max}} = \mathcal{D} + \mathcal{S} = \frac{D}{P_{max}} + \frac{S}{P_{max}}. \tag{2}$$

We then define δ and σ as:

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

In other words, δ and σ denote what fraction of the total power dissipation at maximum frequency is caused by switching activity and what is caused 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:

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

and

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

Combining these equations with Equation (2) then results in:

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

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

$$\mathcal{E}(\mathcal{F}, \mathcal{V}) = \mathcal{P}(\mathcal{F}, \mathcal{V})\mathcal{T} = \delta \mathcal{V}^2 + \sigma \mathcal{V}/\mathcal{F}. \tag{3}$$

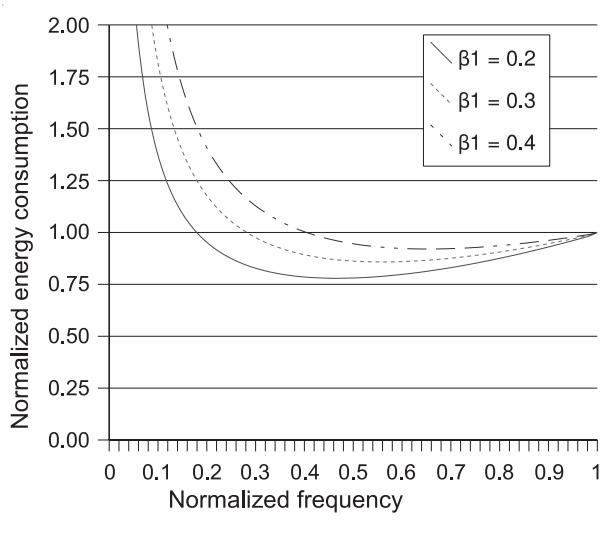


Figure 2. Normalized energy consumption as a function of the normalized frequency.

From this equation it can be seen that voltage scaling only reduces the energy consumption if for a certain $\mathcal{F} < 1$ there exists a $\mathcal{V} < 1$, so that $\mathcal{E}(\mathcal{F}, \mathcal{V}) < 1$.

The supply voltage of a processor must be sufficient 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 clock frequency. From [10], we take the following expression approximating the relation between normalized voltage and frequency:

$$\mathcal{V} = \beta_1 + \beta_2 \mathcal{F}, \tag{4}$$

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

Combining this with Equation (3) yields:

$$\mathcal{E}(\mathcal{F}) = \delta(\beta_1 + \beta_2 \mathcal{F})^2 + \sigma(\beta_1/\mathcal{F} + \beta_2). \tag{5}$$

In Figure 2, the normalized energy consumption is depicted as a function of the normalized frequency for a number of different relative threshold voltages. In this figure, we have used processors where, at maximum frequency, the leakage current is responsible for 50% of the total energy consumption ($\sigma = 0.5$). From this figure, it can be seen that a higher threshold voltage diminishes the possibility to effectively employ voltage scaling.

Figure 2 also shows that, for a certain threshold voltage and amount of leakage current, there is an optimal frequency, at which the total energy consumption is minimized. With a threshold voltage of 0.3 times the maximum supply voltage ($\beta_1 = 0.3$), this optimal frequency is about

0.56 times the maximum frequency. This implies that scaling the frequency to below this point will result in a higher energy consumption than when running the processor at a normalized frequency of 0.56 and turning the processor off for the remainder of time. In order to do this, however, it must be possible to shut the processors down temporarily. This is outside the scope of this work.

3.2 Voltage Scaling in a Multiprocessor Environment

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 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 the 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, so that we have to find a balance between the number of employed processors and the clock frequency.

In our approach, we take the following assumptions: First, we assume that all employed processors must stay on all the time. In other words, it is not possible to turn a processor on or of during execution. Second, we assume that all processors run at the same clock frequency. Furthermore, we consider additional power dissipation and delay caused by communication to be beyond of the scope of this paper. Under these assumptions, the normalized power consumption of a multiprocessor with N processors is given by:

$$\mathcal{P}_{multi} = N(\alpha\delta\mathcal{V}^2\mathcal{F} + \sigma\mathcal{V}), \tag{6}$$

where α denotes the activity (i.e. the fraction of time that the processors are busy). The activity is given by:

$$\alpha = \sum_{v \in V} w(v)/ND,$$

where $w(v)$ denotes the execution time of task v, N the number of processors, and D the deadline. Combining Equation (6) with Equation (4) then results in:

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

4 Multiprocessor Scheduling

Due to static power dissipation, employing the maximum number of processors will not always result in a decreased energy consumption. Our leakage-aware multiprocessor scheduling (LAMPS) algorithm determines the number of processors that results in the lowest energy consumption.

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.

Our leakage-aware multiprocessor scheduling (LAMPS) algorithm works as follows. Let the task graph be represented by a weighted directed acyclic graph (DAG) $G = (V, E, w)$, where V corresponds to the tasks, E to task dependences, and $w(v)$ denotes the execution time of task v.

First we determine the minimal number of processors required to finish the tasks before the deadline. This step is performed as follows. First, we establish a lower bound on the number of processors N_{lwb} needed to complete the tasks before the deadline D and an upper bound on the number of processors N_{upb} that can be employed efficiently:

$$N_{\text{lwb}} = \lceil \sum_{v \in V} w(v)/D \rceil, \quad N_{\text{upb}} = |V|.$$

Thereafter, a binary search is performed on the interval $[N_{\text{lwb}}, N_{\text{upb}}]$ to determine the minimal number of processors required to finish the task graph on time. First, it is determined if $N = (N_{\text{lwb}} + N_{\text{upb}})/2$ are sufficient to finish before the deadline. This is done using a list scheduling algorithm that employs the earliest deadline first (EDF) priority function. If the makespan of the schedule produced by the list scheduler is less than or equal to the deadline, the search continues on the interval $[N_{\text{lwb}}, N]$. If not, the search continues on the interval $[N + 1, N_{\text{upb}}]$.

After having found the minimal number of processors N_{min} required, the number of processors that requires the least amount of power is determined. This step is performed as follows. First, we determine the total power consumption for N_{min} processors. This is done by lowering the clock frequency and supply voltage so that the task graph is completed exactly at the deadline, as in the S&S algorithm. In other words, we stretch the schedule so that it finishes exactly on time. This is also done for $N_{\text{min+1}}$, $N_{\text{min+2}}$, etc. processors, until increasing the number of processors no longer decreases the makespan of the schedule. At this point, increasing the number of processors will aways increase the total power consumption. The algorithm returns the configuration (number of processors) that requires the least amount of power.

In Section 3.1, we noted that scaling the frequency to below the optimal frequency will actually increase the energy consumption. However, since we assume that the option to shut down processors temporarily is not available, results with frequencies below the optimum are still valid.

The reason for performing a linear search instead of a binary search in the second phase of the algorithm is that the power consumption as a function of the number of processors can have local minima. Consequently, a binary search will not always find the optimal solution. An example of this will be given in Section 5.

The time complexity of the algorithm depends on the structure of the task graph and the time it takes to perform list scheduling. Let T_{ls} denote the time required to perform list scheduling. The time T_{LAMPS} taken by the LAMPS algorithm is given by:

$$T_{\text{LAMPS}} = \log_2(N_{\text{upb}} - N_{\text{lwb}}) \cdot T_{\text{ls}} + M \cdot T_{\text{ls}},$$

where M is the number of iterations of the second phase (number of iteration required until the makespan of the generated schedule no longer decreases). In practice, for all benchmarks finding the optimal configuration never took more than six seconds on a 3GHz Pentium 4.

5 Experimental Results

In this section, we present the results of our LAMPS scheduling approach and compare it to the S&S algorithm.

In the experiments, we assume a technology where leakage current contributes to the overall power consumption to a much larger extend than it does today. Specifically, it is assumed that half of the power consumption at maximum frequency is due to this leakage current ($\delta = 0.5, \sigma = 0.5$). Furthermore, we postulate that the threshold voltage is 0.3 times the supply voltage ($\beta_1 = 0.3$, $\beta_2 = 0.7$), which, according to [10], is representative for current technology.

Note that, although we have assumed a relatively high leakage current in this processor, according to Figure 2, it is still theoretically possible to reduce energy consumption by voltage/frequency scaling, until the frequency becomes lower than 29% of the maximum frequency.

The experimental results have been obtained using our in-house scheduling tool and the power model described in Section 3. Table 1 lists the benchmarks that have been used, as well as the number of nodes and edges, the length of the critical path, and the total weight of all the nodes (total work). The first three benchmarks have been derived from real applications, while the other three have been randomly generated. These benchmarks were taken from the *Standard Task Graph Set* [8]. Since this set does not provide deadlines, we have used deadlines of 1.5, 2, 4, and 8 times the critical path length (CPL).

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

Table 1. Employed benchmarks and their main characteristics.

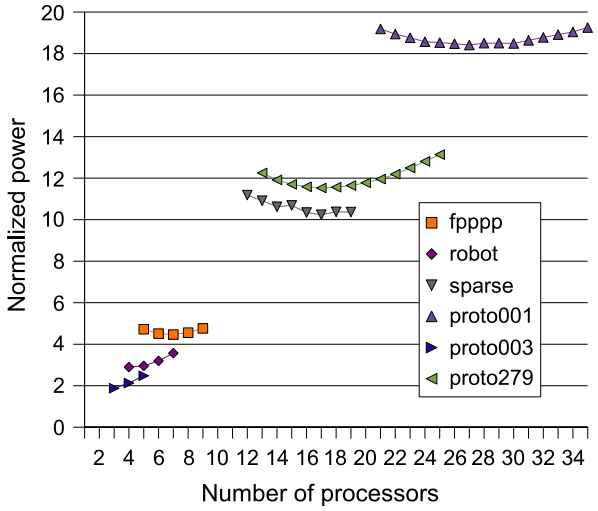


Figure 3. Normalized energy consumption for different benchmarks with the deadline at 1.5 times the critical path length.

Figure 3 depicts the normalized total power consumption as a function of the number of processors employed for the case that the deadline is 1.5x the length of the critical path. From this figure, it can be seen that there are minima that are not globally optimal. This happens, for example, for the *sparse* benchmark at 14 processors. Therefore, a full search must be performed on the number of processors, in order to find the optimum for a certain graph and deadline.

Table 2, 3, 4, and 5 depict the obtained results for the case that the deadline is 1.5, 2, 4, and 8 times as large as the critical path length, respectively. Results are presented for the LAMPS algorithm as well as the S&S algorithm. For each benchmark and scheduling algorithm the (optimal) number of processors N, the normalized frequency $\mathcal{F}$, and the normalized total power consumption $\mathcal{P}$ are listed.

Note that for the S&S algorithm the normalized clock frequency can be derived as the ratio of the critical path

benchmark	LAMPS N	LAMPS $\mathcal{F}$	LAMPS $\mathcal{P}$	S&S N	S&S $\mathcal{F}$	S&S $\mathcal{P}$
fpppp	7	0.76	4.46	9	0.67	4.76
robot	4	0.82	2.90	7	0.67	3.57
sparse	17	0.72	10.24	19	0.67	10.37
proto001	27	0.79	18.41	36	0.67	19.37
proto003	3	0.74	1.88	5	0.67	2.48
proto279	17	0.78	11.53	25	0.67	13.14

Table 2. Results for deadline at 1.5 times the length of the critical path.

benchmark	LAMPS N	LAMPS $\mathcal{F}$	LAMPS $\mathcal{P}$	S&S N	S&S $\mathcal{F}$	S&S $\mathcal{P}$
fpppp	6	0.64	3.19	9	0.50	3.63
robot	3	0.77	2.05	7	0.50	2.75
sparse	14	0.65	7.51	19	0.50	7.84
proto001	22	0.69	13.01	36	0.50	14.68
proto003	2	0.75	1.32	5	0.50	1.93
proto279	15	0.65	8.20	25	0.50	10.04

Table 3. Results for deadline at 2 times the length of the critical path.

benchmark	LAMPS N	LAMPS $\mathcal{F}$	LAMPS $\mathcal{P}$	S&S N	S&S $\mathcal{F}$	S&S $\mathcal{P}$
fpppp	3	0.58	1.47	9	0.25	2.33
robot	2	0.57	0.98	7	0.25	1.79
sparse	7	0.59	3.48	19	0.25	4.96
proto001	12	0.60	6.19	36	0.25	9.35
proto003	1	0.72	0.63	5	0.25	1.27
proto279	8	0.58	3.97	25	0.25	6.45

Table 4. Results for deadline at 4 times the length of the critical path.

benchmark	LAMPS N	LAMPS $\mathcal{F}$	LAMPS $\mathcal{P}$	S&S N	S&S $\mathcal{F}$	S&S $\mathcal{P}$
fpppp	2	0.42	0.75	9	0.12	1.81
robot	1	0.56	0.48	7	0.12	1.40
sparse	3	0.66	1.71	19	0.12	3.83
proto001	6	0.59	3.04	36	0.12	7.24
proto003	1	0.36	0.33	5	0.12	1.00
proto279	4	0.57	1.96	25	0.12	5.01

Table 5. Results for deadline at 8 times the length of the critical path.

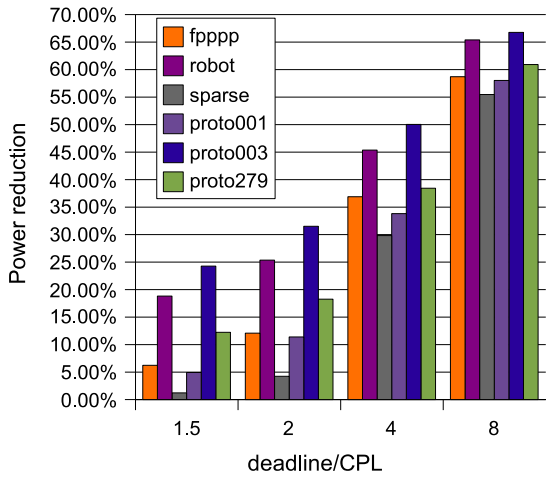


Figure 4. Power reduction achieved by our LAMPS scheduling algorithm over S&S.

length to the deadline. Also note that for S&S, the number of processors is independent of the deadline. This algorithm employs as many processors as possible to finish the tasks as early as possible in order to maximize the amount of slack that can be used to lower the clock frequency. LAMPS, on the other hand, uses fewer processors and a slightly higher clock frequency to balance the amount of static and dynamic power dissipation.

Figure 4 depicts the power reduction achieved by the LAMPS algorithm compared to S&S. Recall that since the schedules generated by both algorithms finish at the same time, they can be compared by power dissipation instead of energy consumption. It can be seen that LAMPS achieves significant energy savings relative to S&S. Furthermore, as expected, the improvement increases with the deadline. For example, if the deadline is tight (1.5x the critical path length), the relative power reduction ranges from 1% to 24%. On the other hand, if the deadline is relatively loose (8x the critical path length), the improvement ranges from approximately 55% to 67%. If the deadline is less strict, fewer processors can be used to finish the tasks on time. This allows LAMPS to improve upon S&S which always employs as many processors as can be used to reduce the makespan of the generated schedule.

It can also be seen from Tables 2 to 5 and Figure 4 that the amount of improvement also depends on the benchmark. For example, for the sparse benchmark a power reduction of only 1% is achieved when the deadline is 1.5x the critical path length, while for proto003 a power saving of 24% is attained. The reason for this behavior is that LAMPS requires 17 processors to finish sparse on time, while S&S requires 19, so only 2 or $2/19 = 10.5\%$ of the processors

can be turned off to reduce power dissipation. On the other hand, for proto003 2 out of 5 or 40% of the processors can be turned off, which results in a more significant power reduction. The geometric means of the savings by LAMPS upon S&S are: 11%, 17%, 39%, and 61% for deadlines of respectively 1.5, 2, 4, and 8 times the length of the critical path.

Generally, processors that support DVS can only scale to a fixed number of predetermined voltage/frequency pairs. In this work, however, we have assumed that the frequency and voltage can be scaled to any value between 0 and the maximum. To show the impact of scaling in discrete steps, we have also performed the experiments with the limitation that the normalized supply voltage can only be scaled in discrete steps of 0.05, similar to [6]. Because we base the deadline on the length of the critical path, the frequency in S&S is solely determined by the deadline. Therefore, the increase in power consumption when using discrete scaling in S&S is independent of the structure of the benchmark. In this case, this increase is completely determined by the distance to the next higher supported frequency and the fraction of time the processors are busy (α). For deadlines of 1.5, 2, 4, and 8 times the length of the critical path, the increases in power consumption for S&S are in the ranges $5.4-5.6\%$, $0-0\%$, $5.6-5.8\%$, and $3.3-3.4\%$ respectively. Because one of the supported normalized frequencies is exactly 0.5, there is no loss when the deadline is set at 2 times the length of the critical path. With LAMPS on the other hand, the operating frequency will vary across the different benchmarks. As a result, the increase in power consumption will be different for different benchmarks. Figure 5 depicts the improvements of LAMPS upon S&S, when scaling the voltage in discrete steps. Because there is no increase in power consumption for schedules produced by S&S with a deadline of 2 times the length of the critical path, the improvements by LAMPS for this deadline are lower than when using continuous scaling. For the other deadlines, the results depend on whether S&S or LAMPS suffers more from having to use discrete steps. If LAMPS has a higher increase in power consumption, the improvements upon S&S compared to the continuous case will be less. Examples of this are robot and proto003 for deadlines of 4 times the length of the critical path. When S&S suffers more from the discretization than LAMPS, on the other hand, the improvements will be higher than in the continuous case. This happens, for example, with all of the benchmarks when deadlines of 1.5 times the length of the critical path are used. In general, we can conclude that LAMPS can also be used to effectively reduce the energy consumption if voltage scaling is limited to discrete steps. It must be noted, however, that these steps should be chosen carefully, in order not to waste too much power.

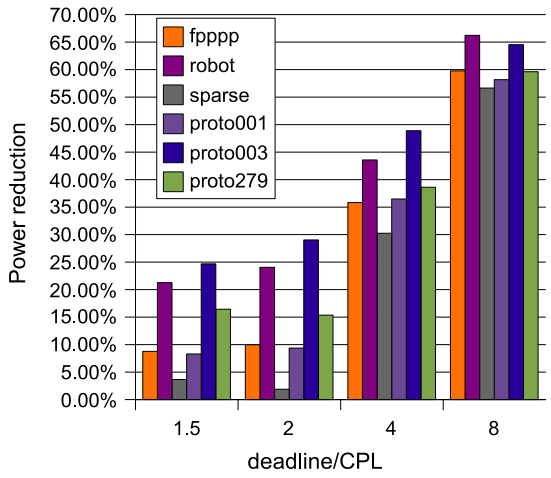


Figure 5. Power reduction achieved by our LAMPS scheduling algorithm over S&S, when scaling the voltage in discrete steps.

6 Conclusions and Future Work

As feature sizes keep decreasing, the contribution of leakage current to the total energy consumption is expected to increase significantly. In this paper we have shown that when the static power dissipation becomes more significant, employing the maximum number of processors to maximize the amount of slack that can be used to lower the supply voltage is no longer beneficial from an energy perspective.

Depending on the amount of leakage current and the amount of parallelism exhibited by the application, the proposed LAMPS algorithm determines the number of processors, their clock frequency, and the corresponding supply voltage that minimizes the total energy consumption. The experimental results show that LAMPS reduces the total amount of dissipated power/energy by up to 24% for tight deadlines and by up to 67% for loose deadlines.

When voltage scaling is limited to discrete steps, the energy consumption of a schedule produced by LAMPS will be slightly higher. However, since this is also the case with S&S, the improvements made by LAMPS will be lower in some cases, while they will be higher in other ones. In general, the results for discrete voltage scaling are close to the results for scaling on a continuous range.

Ultimately, 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 energy solution will be to perform as much as possible sequentially on one processor, and to employ parallelism only if the required performance demands to do so.

We have assumed that all processors operate at the same frequency. By slowing down some processors more than others, it could be possible to produce a more balanced schedule that consumes less power than the schedule generated by LAMPS. Shutting down processors temporarily is an option we also intend to pursue. Finally, we also intend to investigate the influence of communication.

References

[1] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. 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*, pages 518–525, 2004.

[2] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *Proc. Int. Symp. on Computer Architecture*, pages 148–157, 2002.

[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] H. Hofstee. Power Efficient Processor Architecture and the Cell Processor. In *Proc. Int. Symp. on High-Performance Computer Architecture*, pages 258 – 262, 2005.

[6] R. Jejurikar, C. Pereira, and R. Gupta. Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems. In *Proc. Conf. on Design Automation*, pages 275–280, 2004.

[7] N. Jha. Low-Power System Scheduling, Synthesis and Displays. *IEE Proc. on Computers and Digital Techniques*, 152(3):344– 352, 2005.

[8] H. Kasahara, T. Tobita, T. Matsuzawa, and S. Sakaida. Standard Task Graph Set. `http://www.kasahara.elec.waseda.ac.jp/schedule/`.

[9] 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.

[10] N. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan. Leakage Current: Moore's Law Meets Static Power. *IEEE Computer*, 36(12):68–75, 2003.

[11] 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*, pages 721–725, 2002.

[12] P. Stravers and J. Hoogerbrugge. Homogeneous Multiprocessing and the Future of Silicon Design Paradigms. In *Proc. Int. Symp. on VLSI Technology, Systems, and Applications*, 2001.

[13] G. Varatkar and R. Marculescu. Communication-Aware Task Scheduling and Voltage Selection for Total Systems Energy Minimization. In *Proc. Int. Conf. on Computer-Aided Design*, page 510, 2003.

[14] L. Yan, J. Luo, and N. 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, 2003.

[15] Y. Zhang, X. Hu, and D. Chen. Task Scheduling and Voltage Selection for Energy Minimization. In *Proc. Conf. on Design Automation*, pages 183–188, 2002.

[16] D. Zhu, R. Melhem, and B. 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.