J Sign Process Syst
DOI 10.1007/s11265-008-0176-8

Leakage-Aware Multiprocessor Scheduling

Pepijn de Langen · Ben Juurlink

Received: 12 October 2007 / Revised: 4 January 2008 / Accepted: 13 March 2008

Abstract When peak performance is unnecessary, Dynamic Voltage Scaling (DVS) can be used to reduce the dynamic power consumption of embedded multiprocessors. In future technologies, however, static power consumption due to leakage current is expected to increase significantly. Then it will be more effective to limit the number of processors employed (i.e., turn some of them off), or to use a combination of DVS and processor shutdown. In this paper, leakage-aware scheduling heuristics are presented that determine the best trade-off between these three techniques: DVS, processor shutdown, and finding the optimal number of processors. Experimental results obtained using a public benchmark set of task graphs and real parallel applications show that our approach reduces the total energy consumption by up to 46% for tight deadlines (1.5× the critical path length) and by up to 73% for loose deadlines (8× the critical path length) compared to an approach that only employs DVS. We also compare the energy consumed by our scheduling algorithms to two absolute lower bounds, one for the case where all processors continuously run at the same frequency, and one for the case where the processors can run at different frequencies and these frequencies may change over time. The results show that the energy reduction achieved by our best approach is close to these theoretical limits.

Keywords Multiprocessor · Scheduling · Leakage power · Voltage scaling

1 Introduction

Recently, (single-chip) multiprocessors such as the ARM11 MPCore [4] and the IBM/Sony/Toshiba Cell architecture [5] have been introduced on the high-performance embedded market. The power consumption of such systems is a prime design consideration. It consists of a dynamic part (due to switching activity) and a static part (due to leakage current). In past technologies, the dynamic power was much larger than the static power. With each technology generation, however, the leakage current is predicted to increase by a factor of five [6] and the static power consumption is predicted to surpass the dynamic power consumption [7].

To reduce power consumption, many techniques have been proposed such as shutting down unused parts [8] or to support multiple supply voltages [9]. In this paper we consider the problem of scheduling tasks on a real-time multiprocessor system that supports these hardware techniques. The goal of the proposed scheduling algorithms is to minimize the total energy consumption.

To guarantee real-time performance, embedded multiprocessors and Systems-on-Chip in general are

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

This paper is a revised and extended version of the work that appeared in [2, 3].

P. de Langen (✉) · B. Juurlink
Faculty of Electrical Engineering, Mathematics and Computer Science, Computer Engineering Lab., Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
e-mail: pepijn@ce.et.tudelft.nl

B. Juurlink
e-mail: benj@ce.et.tudelft.nl

usually overbudgeted, e.g., they generally contain more processing cores than needed. When the dynamic power is much larger than the static power, an effective technique to reduce the energy is to schedule the tasks on as many processors as possible to reduce the makespan of the schedule. Thereafter, the remaining time before the deadline (the *slack*) is used to scale down the supply voltages and operating frequencies. We refer to this technique as *Schedule-and-Stretch* (S&S). When the static and dynamic power are comparable, however, S&S is no longer effective because it increases the leakage current by using more processors than necessary and by lengthening the time it takes to complete the computation.

Scheduling heuristics are presented that improve upon S&S by determining the optimal trade-off between Dynamic Voltage Scaling (DVS) and processor shutdown (PS). The first algorithm, called Leakage-Aware MultiProcessor Scheduling (LAMPS), does not employ as many processors as possible to maximize the amount of slack that can be used to lower the supply voltage. Instead, it determines an optimal balance between the number of processors that should be used and the level of voltage/frequency scaling. We then extend both S&S and LAMPS with the option to put processors temporarily in a deep sleep or shutdown mode. This technique is referred to as PS, and hence, these strategies are referred to as S&S+PS and LAMPS+PS, respectively.

Furthermore, we formulate two absolute lower bounds that produce schedules that consume the least amount of energy possible. The first is for the case where all processors run at the same frequency throughout the entire schedule. The schedules produced by S&S(+PS) and LAMPS(+PS) have this property. The second is for the case where the processors can run at different frequencies and these frequencies may change over time.

Experimental results are obtained using a public benchmark set of task graphs with precedence constraints and real parallel applications. The results show that our best approach (LAMPS+PS) reduces the total energy consumption by up to 46% for tight deadlines (1.5× the critical path length) and by up to 73% for loose deadlines (8× the critical path length (CPL)) compared to S&S. Compared to LAMPS, LAMPS+PS decreases the total energy consumption by up to 12% respectively 18%. We also analyze how the results are affected by the average amount of parallelism, which is defined as the total amount of work divided by the CPL. Comparing the results to the theoretical lower bounds indicates there is little room left for improvement. For example, for fairly coarse-grain task graphs LAMPS+PS attains over 94% of the possible energy saving, provided the frequency is the same for all active processors and constant throughout the schedule.

This paper is organized as follows. Section 2 contains an overview of related work. The system and application model, the power model, and the DVS and PS techniques are explained in detail in Section 3. Section 4 reviews S&S and presents LAMPS and extends both of them with the option to shut down processors temporarily. Experimental results for randomly generated as well as task graphs derived from real applications are provided in Section 5. Finally, in Section 6, conclusions are drawn and some directions for future research are given.

2 Related Work

Reducing power consumption has been an important research topic in recent years and many techniques at the process, circuit design, and micro-architectural level have been proposed. One of the most promising techniques is DVS, where both the clock frequency and the supply voltage are scaled down when peak performance is not needed. DVS is also referred to as dynamic voltage/frequency scaling. Several existing processors such as the Intel XScale [10] support DVS.

Applying DVS to multiprocessor scheduling has been investigated by a significant number of researchers. An overview is provided by Jha [11]. As described in Section 1, one approach is to use an existing scheduling algorithm, such as list scheduling with earliest deadline first (EDF), to finish the tasks as early as possible and to use the remaining slack before the deadline to lower the supply voltage. This technique has been proposed by several authors [1, 12] using different names and, therefore, we refer to it as (S&S). Leakage current was not included in their power models, however.

Jejurikar et al. [13] presented a detailed power model that includes static as well as dynamic power. We use the same power model. They further showed that there is an optimal operating point, called the critical speed, at which the total energy consumption is minimized. Lowering the supply voltage below this point increases the energy consumption. They computed processor slowdown factors based on the critical speed. A similar approach was followed by [14], who employed a fixed priority instead of EDF. In contrast to our work, both these works focussed on single-processor scheduling and assumed that tasks are independent and arrive periodically with deadlines. The same real-time model was assumed in [15], but DVS was not considered. Irani et al. [16] also used this model but assumed a continuous voltage range and presented a theoretical analysis of systems which can use DVS and PS. Specifically, they presented an offline algorithm with a competitive ratio of 3 and an online algorithm with a constant competitive ratio.

Zhang et al. [17] used the same real-time model as we do (weighted directed acyclic graphs (DAGs) with deadlines). They did not use EDF scheduling but scheduled in such a way to have more slowdown opportunities. In this paper we analyze the effect of employing a different scheduling algorithm. Furthermore, they did not attempt to determine the number of processors that yields the least energy consumption. Kianzad et al. [18] presented an integrated approach, combining scheduling and DVS in a genetic algorithm. However, they did not consider PS. Varatkar et al. [19] proposed to execute part of the code on a lower supply voltage while minimizing communication. Some researchers proposed to improve DVS by also adjusting the threshold voltage when scaling the supply voltage [20, 21]. Others extended this to scheduling for real-time multiprocessor systems [22, 23]. None of these works, however, attempted to determine the optimal number of processors.

Xu et al. [24] proposed to minimize energy consumption by both DVS and choosing the correct number of employed processors. Their work, however, targets embedded clusters in which the nodes provide the same type of service in a client-server model. Furthermore, these authors do not consider static scheduling but instead propose an online algorithm similar to [1].

Our work differs in the following ways. First, we assume that applications are represented as weighted DAGs whereas many others assumed independent periodic tasks with deadlines. Second, we focus on multiprocessor scheduling while others focussed mainly on single-processor scheduling. Third, we use a detailed power model and limit the voltage scaling to discrete steps. Fourth, we exploit DVS and PS as well as finding the optimal number of processors. Finally, we use a publicly available set of task graphs and a task graph derived from a real application (MPEG-1), whereas most others used randomly generated graphs.

3 Preliminaries

In this section we describe the system and application models, the power model, as well as two primary ways to reduce power dissipation: DVS and PS.

3.1 System and Application Model

We assume a shared memory multiprocessor system running parallel applications, for which the scheduling and mapping are statically determined. The applications are represented as weighted DAGs, where nodes correspond to tasks, edges to task dependences, and node weights to task processing times. We furthermore assume that this system is CPU bound. As explained by Liberato et al. [25], real-time applications with periodic tasks can be translated to DAGs using the *frame-based scheduling* paradigm.

Another common application model based on functional or pipelining parallelism is Kahn Process Networks [26], where a group of processes are connected by communication channels to form a network of processes. Each process is in principle infinite and receives data over its input channels, processes it, and sends the results over the output channels. Here there is not a single deadline but a certain throughput must be guaranteed. This model can be converted to DAGs by making several copies of the KPN, by translating edges in the KPN to edges between successive copies in the DAG and adding an edge from each node in the ith copy to the corresponding node in the $(i+1)$st copy. The output nodes of the first copy are assigned an arbitrary but reasonable deadline. The deadline of the output nodes of each successive copy is set to the deadline of the corresponding node in the previous copy plus the reciprocal of the throughput. A simple example is depicted in Fig. 1. In the KPN in Fig. 1a, task T_1 successively receives inputs $I_1, I_2, \ldots,$ processes

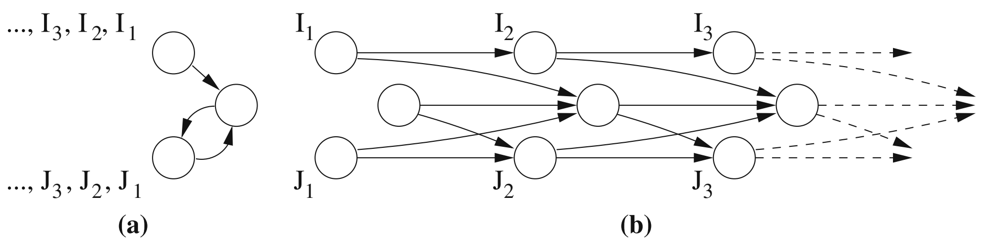


Figure 1 Simple example for translating KPNs into DAGs (**a**, **b**).

them, and sends the results to T_2. Task T_3 receives inputs $J_1, J_2, \ldots$ but also receives data from T_2. It combines input J_{i+1} with the ith data received from T_2. In the DAG in Fig. 1b, each node is replicated a number of times. Let T_i^j denote the jth copy of task T_i. Then T_1^j receives input I_j and T_3^j receives input J_j. There are edges from T_1^j and T_3^j to T_2^{j+1}. Because T_3 combines input J_{i+1} with the ith data received from T_2, there are also edges from T_2^j to T_3^{j+1}. To indicate that not all inputs are available at time zero, there are also edges from T_i^j to T_i^{j+1}. This could also be modeled by adding dummy input nodes whose weights are equal to the time the input becomes available.

Mainly due to unpredictable behavior in the memory system, the execution time of a task does not solely depend on the clock frequency. However, since reducing the frequency will make memory accesses relatively less costly, it is safe to assume that executing a task on $1/N^{th}$ of the frequency will take at most N times as much time.

3.2 Power Model

We use the power model described in [13], which in turn is based on the model and parameters given in [21], where it has been verified with SPICE simulations. In this model, the power consumption of a processor is given by:

$$P = P_{\mathrm{AC}} + P_{\mathrm{DC}} + P_{\mathrm{on}},$$

where P_{AC} is the dynamic power consumption (due to switching activity), P_{DC} is the static power consumption (due to leakage current), and P_{on} is the intrinsic power consumption needed to keep the processor on. Like [13], we assume P_{on} is $0.1W$. The dynamic power is given by:

$$P_{\mathrm{AC}} = aC_{\mathrm{eff}} V_{\mathrm{dd}}^2 f,$$

where a is the activity factor, C_{eff} is the effective switching capacitance, V_{dd} is the supply voltage, and f is the operating frequency. The static power is given by:

$$P_{\mathrm{DC}} = V_{\mathrm{dd}} I_{\mathrm{subn}} + |V_{\mathrm{bs}}|\, I_j,$$

where I_{subn} is the sub-threshold leakage current, V_{bs} is the voltage applied between body and source, and I_j is the reverse bias junction current. The sub-threshold leakage current is given by:

$$I_{\mathrm{subn}} = K_3 e^{K_4 V_{\mathrm{dd}}} e^{K_5 V_{\mathrm{bs}}},$$

Table 1 Constants for 70 nm technology ([13, 21]).

Constant	Value
K_1	0.063
K_2	0.153
K_3	$5.38 \cdot 10^{-7}$
K_4	1.83
K_5	4.19
K_6	$5.26 \cdot 10^{-12}$
K_7	-0.144
V_{dd0}	1.0
V_{bs}	-0.7
α	1.5
V_{th1}	0.244
I_j	$4.8 \cdot 10^{-10}$
C_{eff}	$0.43 \cdot 10^{-9}$
L_d	37.0
L_g	$4.0 \cdot 10^6$

where K_3, K_4, and K_5 are constants. The relation between operating frequency, supply voltage, and threshold voltage is:

$$f = (V_{\mathrm{dd}} - V_{\mathrm{th}})^{\alpha} / L_d K_6,$$

where L_d represents the logic depth and K_6 and α are constants for a certain technology. Finally, the threshold voltage is given by:

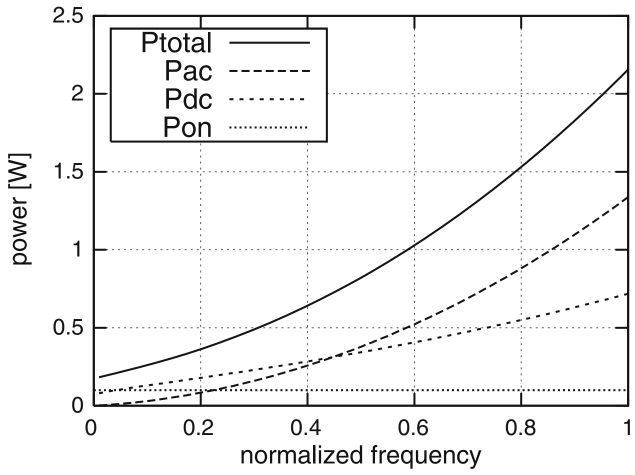
$$V_{\mathrm{th}} = V_{\mathrm{th1}} - K_1 V_{\mathrm{dd}} - K_2 V_{\mathrm{bs}},$$

where V_{th1}, K_1, and K_2 are constants. We use the same $70nm$ technology constants as [13, 21]. These constants are listed in Table 1. The maximum frequency of this processor is 3.1GHz, which requires a supply voltage of 1V. Figure 2a and b depict the resulting power consumption and energy per cycle as a function of the normalized operating frequency.

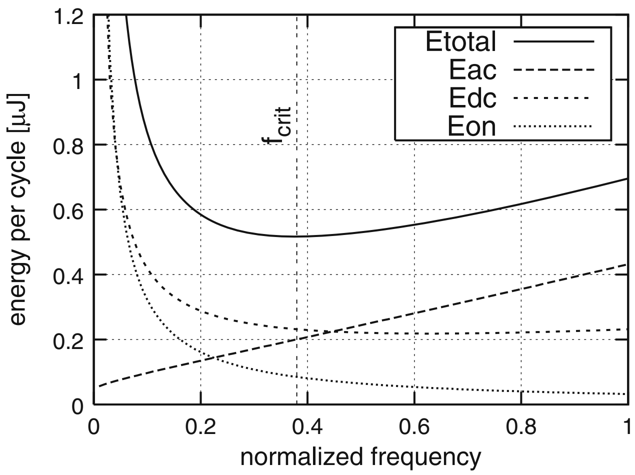
3.3 Dynamic Voltage Scaling

DVS mainly reduces the dynamic power consumption, which increases quadratically with the supply voltage. The static component, although having a exponential relation with supply voltage, does not decrease as much with decreasing supply voltage as the dynamic component, as is depicted in Fig. 2.

Since energy equals power times time, the energy consumption will actually start to increase if the frequency is decreased below a certain point. Figure 2 depicts the energy per cycle as a function of the normalized frequency. It can be seen that the *optimal* or *critical* frequency (f_{crit}) is 0.38 times the maximum. Because of the discrete voltage levels, however, the critical frequency is reached at a supply voltage of 0.7V, corresponding to a normalized frequency of 0.41.



(a)



(b)

Figure 2 Power and energy consumption as a function of the normalized frequency. (**a**) Power consumption. (**b**) Energy consumption.

Scaling below this frequency will reduce the power consumption but not the total energy consumption, provided that the processors can be shut down for the remaining time. When there is no sleep/shutdown mode, scaling below f_{crit} will, in fact, reduce the total energy consumption, since the processors also consume energy for the remaining time.

3.4 Processor Shutdown

The second technique to reduce the energy consumption of a multiprocessor system is to put idle processors temporarily in a deep sleep or shutdown mode. The advantage of this technique over DVS is that it reduces all terms of the total power consumption, not only the dynamic part. When shutting down a processor, however, the contents of, e.g., caches and branch predictors are lost. When a processor is switched back on, they have to be warmed up again, which causes additional delay and consumes extra energy. We use the estimates of Jejurikar et al. [13], who estimated that a processor in sleep state consumes about 50μW of power and that shutting down and resuming a processor incurs an energy overhead of 483μJ. This overhead includes the supply voltage switching as well as the energy spent to warm up caches and predictors. The additional delay incurred by powering down can be hidden by waking up the processor a short time before the end of the idle period.

PS is only beneficial if a processor is idle for a sufficiently long period. Figure 3 depicts this minimum number of idle cycles as a function of the normalized frequency. From this figure it can be seen that, in order to save energy consumption by putting a processor temporarily in shutdown mode, a significant number of idle cycles is required. When clocked at half the maximum frequency, for example, an idle period of at least 1.7 million cycles is required. Since in most cases applications with rather fine-grain tasks will have relatively short idle periods (unless the task graph is very unbalanced), such applications will in general not benefit from to shutting down processors temporarily between the execution of two tasks. However, it might still be energy efficient to shut down at the end of the schedule, provided the deadline is relatively long.

4 Scheduling for Energy Minimization

In this section we review S&S and LAMPS and enhance both scheduling approaches with the option to shut down processors temporarily. In the schedules produced by these approaches, all processors run at the same operating frequency and this frequency is constant throughout the whole schedule. Both S&S

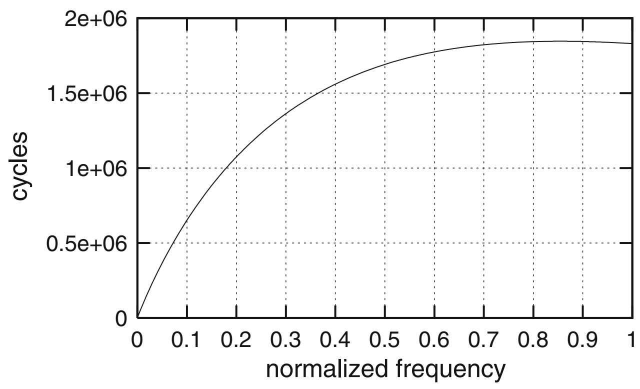


Figure 3 Minimum number of idle cycles required for PS to be beneficial, as a function of the normalized processor frequency.

and LAMPS employ list scheduling with earliest deadline first (LS-EDF) to perform the actual scheduling. EDF does not necessarily produce the best schedule, however. To investigate if other scheduling algorithms could result in additional energy gains, we also present an ideal model in which idle processors are assumed to consume no energy. Furthermore, we also show the improvements that could be attained if the frequency could vary among processors and over time.

4.1 Schedule & Stretch

Figure 4 illustrates the concept behind S&S. The task graph in Fig. 4a is first scheduled using LS-EDF to minimize the makespan of the schedule, or in other words, maximize the amount of slack before the deadline. This is depicted in Fig. 4b, which shows that after the scheduling process, there are certain periods in which a processor is idle. Thereafter, the slack that remains at the end of the schedule is used to lower the clock frequency and supply voltage of all processors, as depicted in Fig. 4c. S&S already reduces the energy consumption by 30% for tight deadlines and by more than 70% for loose ones [2].

4.2 Leakage Aware Multiprocessor Scheduling

In LAMPS, a trade-off is made between the number of processors that should be employed and the amount of voltage scaling. The remaining processors are turned off. The number of processors that minimizes the energy is found as follows. Let the task graph be represented by an acyclic graph $G = (V, E, w)$, 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:

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

Thereafter, a binary search is performed on the interval $[N_{\mathrm{lwb}}, N_{\mathrm{upb}}]$ to determine the minimal number of processors required to finish the task graph on time. First, it is determined if $N = (N_{\mathrm{lwb}} + N_{\mathrm{upb}})/2$ are sufficient to finish before the deadline. This is done using LS-EDF. 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 $[N_{\mathrm{lwb}}, N]$. If not, the search continues on the interval $[N + 1, N_{\mathrm{upb}}]$.

After having found the minimal number of processors N_{min} required, the number of processors that dissipates 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

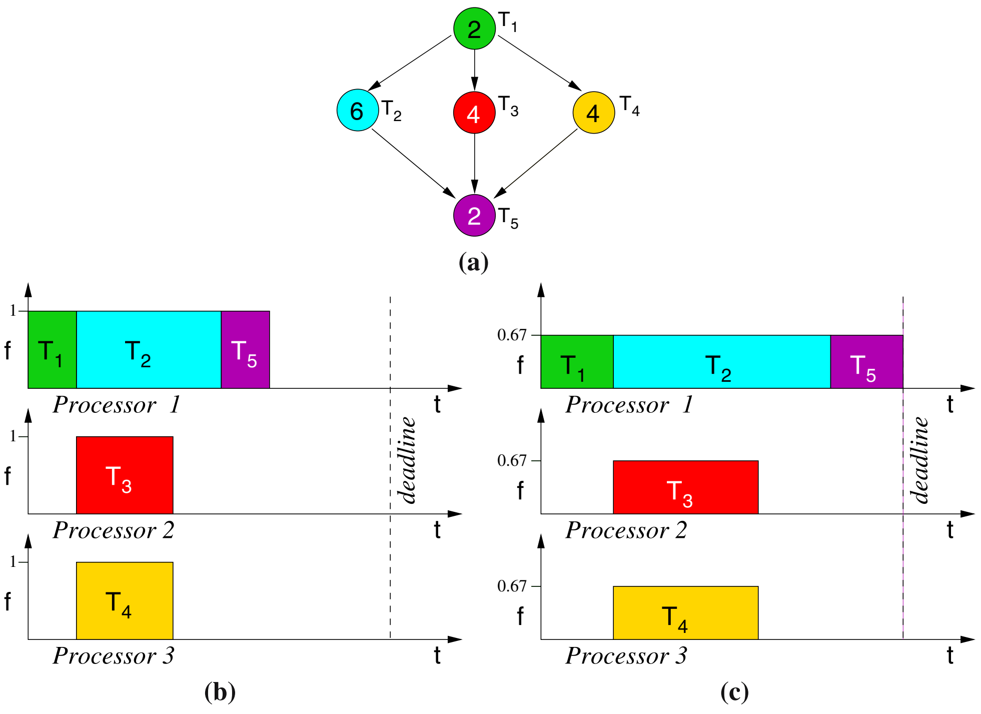


Figure 4 Illustration of S&S. (**a**) Task graph. (**b**) Schedule produced by EDF. (**c**) Schedule produced by S&S.

Figure 5 Pseudocode for the LAMPS heuristic.

```
$Nmin \Leftarrow$ findMinimumProcs($N_{lwb}$, $N_{upb}$)
$numProcessors \Leftarrow N_{min}$
$M \Leftarrow \infty$
$E_{min} \Leftarrow \infty$
**while** true **do**
   $S \Leftarrow$ scheduleGraph($G$, $numProcessors$)
   **if** $M =$ makespan($S$) **then**
      **break**                          /* exit if the makespan is not reduced */
   **end if**
   $M \Leftarrow$ makespan($S$)
   $f_{min} \Leftarrow \lceil M \cdot f_{max}/deadline \rceil$        /* rounded up to the next discrete voltage level */
   $E \Leftarrow$ calculateEnergyConsumption($S$, $f_{min}$)
   **if** $E < E_{min}$ **then**
      $E_{min} \Leftarrow E$
   **end if**
   $numProcessors \Leftarrow numProcessors + 1$
**end while**
```

voltage so that the task graph is completed as close a possible to 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 always increase the total power consumption. The algorithm returns the configuration (number of processors) that dissipates the least amount of energy. Figure 5 depicts the pseudocode for the LAMPS heuristic.

The reason for performing a linear search instead of a binary search in the second phase of the algorithm is that the energy consumption as a function of the number of processors can have local minima. Consequently, a binary search will not always find the optimal solution. Figure 6 depicts the normalized total energy consumption as a function of the number of processors employed for the case that the deadline is 1.5× the length of the critical path. (The benchmarks will be described in Section 5.1.) It can be seen that there are minima that are not global minima. 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.

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 iterations required until the makespan of the generated schedule no longer decreases). In practice, for all benchmarks finding the optimal configuration never took more than 20 seconds on a 3GHz Pentium 4.

Figure 7a illustrates the schedule generated by LAMPS. Instead of 3 processors, the task graph shown

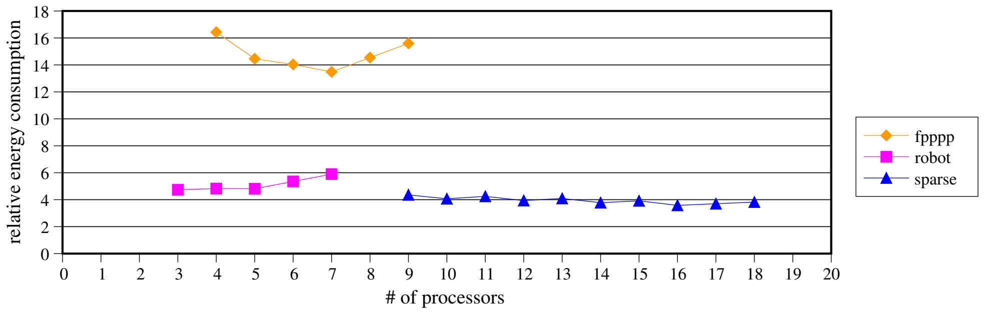


Figure 6 Normalized energy consumption for different benchmarks with the deadline at **2** times the CPL.

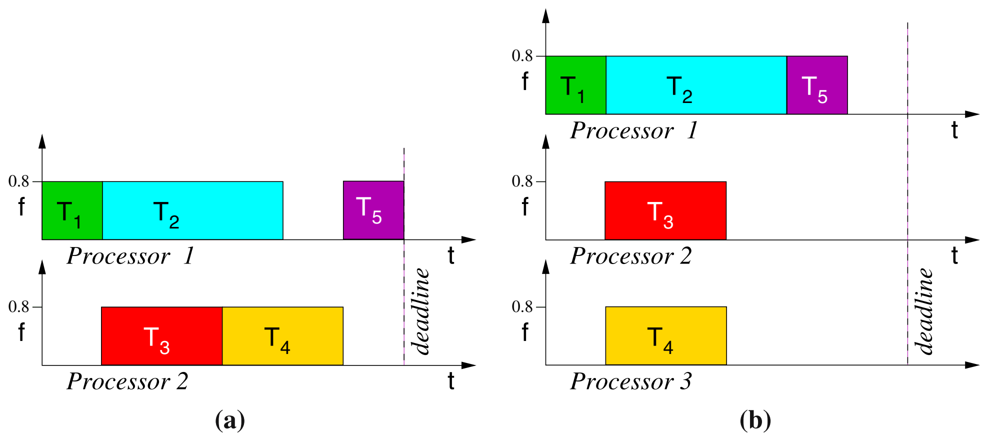


Figure 7 Illustration of LAMPS and S&S+PS. (**a**) Schedule produced by LAMPS. (**b**) Schedule produced by S&S+PS.

in Fig. 4b is scheduled on only 2 processors but with a higher frequency. Nevertheless, because the third processor is turned off, the schedule produced by LAMPS consumes less energy than the schedule generated by S&S.

4.3 S&S+PS and LAMPS+PS

We extend S&S with the option to shut down processors temporarily. We refer to this heuristic as *S&S with PS* (S&S+PS). In S&S+PS, the task graph is again first scheduled using the EDF policy. Thereafter, the optimal balance between processor slowdown (through DVS) and shutdown is determined by gradually scaling the operating frequency from the maximum frequency to the minimum frequency required to meet the deadline using discrete voltage level steps of 0.05 V. For each frequency, the remaining slack both inside as well as at the end of the schedule is used to shut down processors, provided the idle period is longer than the minimum idle period to result in energy savings (cf. Fig. 3). In other words, the slack is only used to shut down a processor if it is large enough to make up for the additional energy consumption due to loss of state.

Figure 7b illustrates the schedule produced by S&S+PS. In this example, only part of the slack at the end of the schedule is exploited to lower the frequency. The remaining slack is used to shutdown processors

Figure 8 Pseudocode for the LAMPS+PS heuristic.

```
$Nmin \Leftarrow$ findMinimumProcs($N_{lwb}$, $N_{upb}$)
$numProcessors \Leftarrow N_{min}$
$M \Leftarrow \infty$
$E_{min} \Leftarrow \infty$
while true do
    $S \Leftarrow$ scheduleGraph($G$, $numProcessors$)
    if $M =$ makespan($S$) then
        break                         /* exit if the makespan is not reduced */
    end if
    $M \Leftarrow$ makespan($S$)
    $f_{min} \Leftarrow \lceil M \cdot f_{max}/deadline \rceil$     /* rounded up to the next discrete voltage level */
    for $f = f_{max}$ to $f_{min}$ do
        $S' \Leftarrow$ processShutdownPeriods($S$, $f$)
        $E \Leftarrow$ calculateEnergyConsumption($S'$, $f$)
        if $E < E_{min}$ then
            $E_{min} \Leftarrow E$
        end if
        $numProcessors \Leftarrow numProcessors + 1$
    end for
end while
```

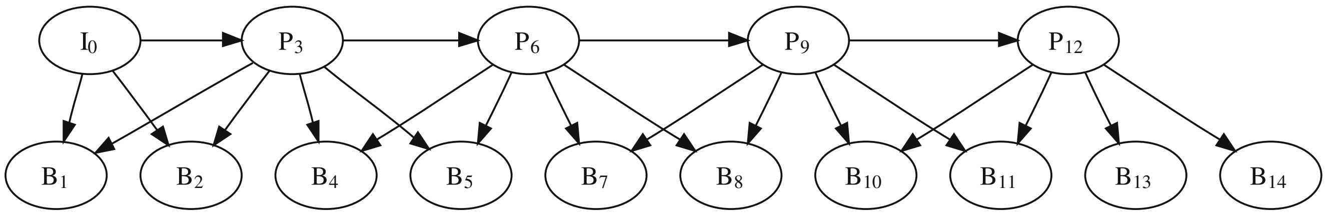


Figure 9 Dependence graph for processing 15 MPEG-1 frames, assuming execution times of 36700900, 178259300, and 73401800 cycles for I, B, and P frames respectively.

temporarily. We note that this example is merely meant for illustration. In reality, for the given task graph it is not advantageous to employ PS.

We remark that the time taken by the scheduling algorithm can be improved by scaling down the frequency to the critical frequency (or the minimum frequency required to meet the deadline if this minimum frequency is larger than the critical frequency), and to use the remaining slack to shut down processors. However, because the effectiveness of PS depends on both the time a processor is idle and on the intrinsic power needed to keep the processor on, such an approach would not find the schedule that minimizes the total energy consumption.

We also enhance the LAMPS heuristic with the option to shut down processors and refer to the resulting heuristic as LAMPS+PS. As in LAMPS, the number of processors that minimizes the total energy consumption is determined by calculating the energy consumption for $N_{\min}$, $N_{\min}+1, \ldots, N_{\max}$ processors, where $N_{\min}$ is the minimal number of processors needed to meet the deadline and $N_{\max}$ is the number of processors that can be employed efficiently. For each number of processors, we then determine the balance between DVS and PS by scaling the frequency from the maximum to the minimum frequency required to meet the deadline. For each frequency, we then use the available slack to shut down processors, similar to the S&S+PS heuristic. The pseudocode for the LAMPS+PS heuristic is depicted in Fig. 8.

4.4 LIMIT-SF & LIMIT-MF

In the approaches described above, the schedule is always produced by EDF. It is known, however, that EDF is not always optimal for multiprocessor scheduling. Furthermore, in our approaches the frequency is always constant throughout the entire schedule. To investigate if additional energy can be saved by employing a different scheduling algorithm or by allowing different frequencies, we also define two lower bounds, one for the case with a single frequency (LIMIT-SF) and one for the case where multiple frequencies are allowed (LIMIT-MF).

LIMIT-SF has the following characteristics. First, idle processors are assumed to consume no energy at all. In other words, only active cycles are considered when calculating the energy consumption and, consequently, there is no benefit from or penalty for shutting down processors. Second, the number of processors is equal to the number of tasks. Since idle processors consume no energy, using fewer processors will not reduce the energy. Third, the frequency is scaled down to the optimal frequency if possible to meet the deadline, or otherwise as much as possible. No schedule can

Table 2 Employed benchmarks from STG [27] and their main characteristics.

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
50	50	66–926	24–447	204–644
100	100	138–1898	29–569	458–1347
300	300	412–8991	45–1164	1517–3568
500	500	698–24497	67–1941	2563–5530
1000	1000	1378–99164	50–3298	5179–11138
2000	2000	2797–396760	48–6770	10563–21615
5000	5000	7132–2491411	62–17386	27009–54010

consume less energy than this ideal model, provided that the frequency is the same for all active processors and is constant throughout the schedule.

The difference between LIMIT-MF and LIMIT-SF is that in LIMIT-MF all tasks are scheduled at the critical frequency. Because of this and since idle processors are assumed to consume no energy, LIMIT-MF is an absolute lower bound, even for the case where processors can run at different speeds and where the frequency may change over time. We note, however, that it may happen that the schedule produced by LIMIT-MF does not meet the deadline.

Since both LIMIT-SF and LIMIT-MF do not depend on any particular scheduling algorithm, this implies that these results cannot be improved by employing a different scheduling algorithm than EDF.

5 Experimental Evaluation

In this section, we present and compare the results of the different scheduling approaches. We use the same power model as used by [21] and [13], as explained in Section 3. We again emphasize that a processor in sleep state consumes 50 μW and that shutting down and waking up a processor dissipates 483 μJ of energy.

5.1 Experimental Setup

For the experiments we use task graphs from the *Standard Task Graph Set* [27], as well as a task graph for MPEG-1 encoding presented by Zhu et al. [1]. The MPEG-1 encoding task graph consists of an encoding sequence of 15 I, B, and P frames, and is depicted in Fig. 9. We have used the maximum execution times for the *Tennis* sequence as presented in [1], scaled to match the maximum clock frequency of 3.1GHz. The deadline was set at 0.5 seconds for a GOP of 15 frames, to match a real-time encoding requirement of 30 frames per second.

The Standard Task Graph Set provides 3 graphs that were generated from actual applications: ‘fpppp’, ‘robot’, and ‘sparse’. This set also contains 2700 randomly generated graphs, grouped by the number of nodes. Each group in this set consists of 180 different graphs. Since the results for differently sized graphs are comparable, we only present the results for 50, 100, 500, 1000, 2000, and 5000 nodes in this work. For both the real application graphs and the groups of random graphs, the number of nodes and edges, the CPL, and the sum of all node weights (total work) are listed in Table 2.

Since the Standard Task Graph Set does not provide deadlines, we use deadlines of 1.5, 2, 4, and 8 times the CPL when running at the maximum frequency of 3.1 GHz. It also does not define the unit of the task weights. Instead, the weights are given as integers in the range from 1 to 300. Therefore, two different scenarios are considered. In the first scenario, corresponding to rather coarse-grain tasks, a weight of 1 in a task graph implies an execution time of $3.1 \cdot 10^6$ cycles, which is 1 millisecond when running at the maximum frequency of 3.1GHz. In the second scenario, corresponding to relatively fine-grain tasks, the same weight implies an execution time of $3.1 \cdot 10^4$ cycles, which at maximum frequency takes 10 microseconds.

5.2 Results for the Standard Task Graph Set

Figures 10 and 11 depict the relative energy consumption for coarse grain and fine grain tasks, respectively. For each scenario, we show the energy consumption for deadlines of 1.5, 2, 4, and 8 times the CPL. Each figure shows the results of the four different approaches explained in Section 4, as well as the theoretical limits. Throughout this section, S&S is used as the baseline against which we compare the other heuristics.

First we compare the energy consumption of the schedules produced by LAMPS to the energy consumption of the schedules generated by S&S. Figures 10 and 11 show that LAMPS improves upon S&S mainly for less strict deadlines. This can be expected because for tight deadlines (1.5$\times$ the CPL), LAMPS requires the same or nearly the same number of processors as S&S to meet the deadline, and therefore consumes the same or nearly the same amount of energy as S&S. In other words, if the deadline is tight, there is less opportunity to turn off processors. For loose deadlines (8$\times$ the CPL), on the other hand, LAMPS consumes significantly less energy than S&S, simply because it can employ fewer processors. In this case LAMPS reduces the total energy consumption by 45% on average compared to S&S with a maximum of 67%. For fine-grain tasks, depicted in Fig. 11, the relative differences between S&S and LAMPS are the same as with coarse-grain tasks, since both heuristics do not shut down processors.

We now compare S&S+PS to S&S. Because S&S employs a large number of processors, it consumes a significant amount of static power. Therefore, S&S+PS improves upon S&S significantly, by shutting down idle processors temporarily. The gains, in this case, are considerably larger for coarse-grain tasks (23% on average with a deadline of 2$\times$ the CPL) than for fine-grain tasks (4% on average with a deadline of 2$\times$ the CPL),

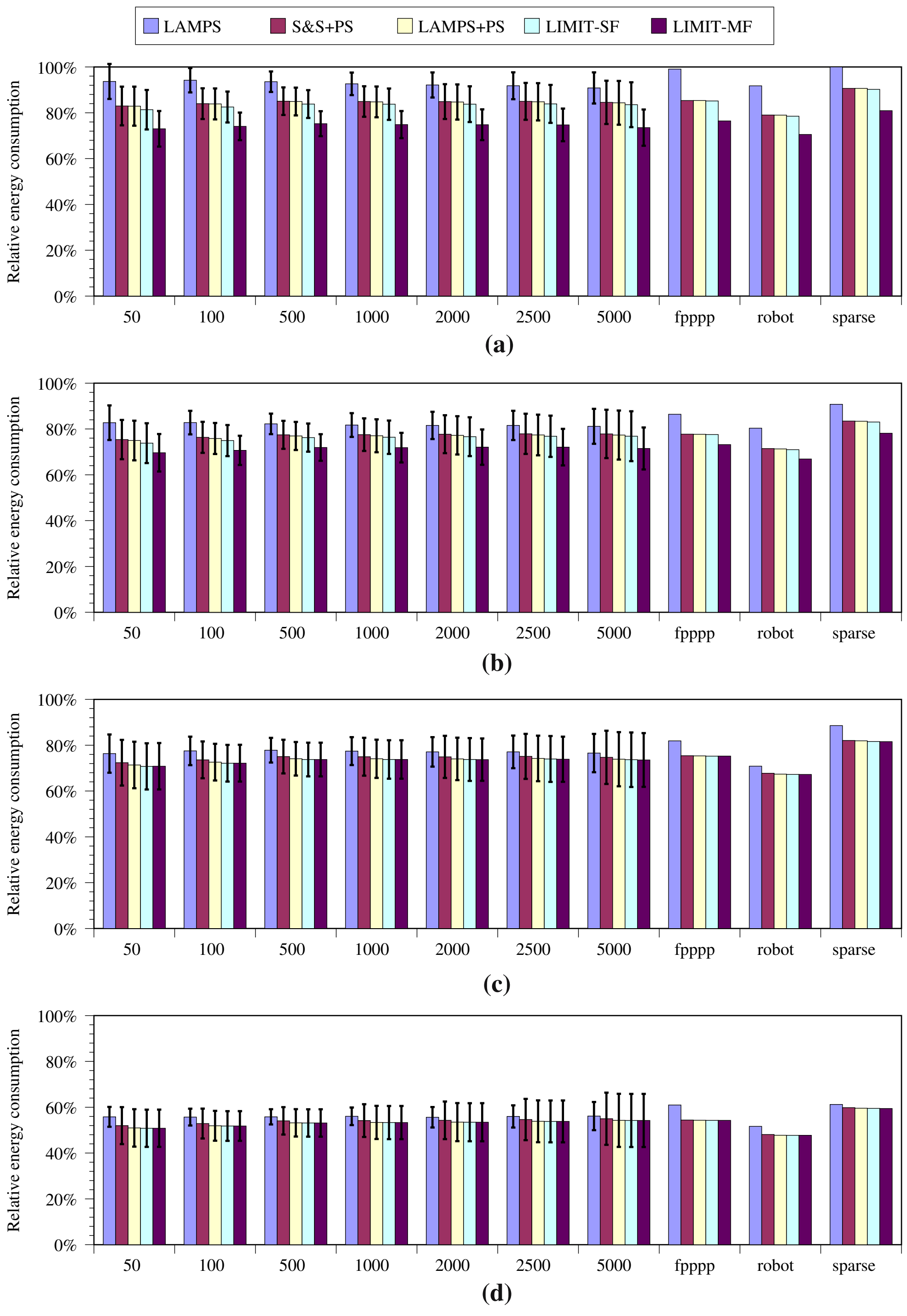


Figure 10 Energy consumption for coarse-grain tasks. (**a**) Deadline = $1.5 \times CPL$. (**b**) Deadline = $2 \times CPL$. (**c**) Deadline = $4 \times CPL$. (**d**) Deadline = $8 \times CPL$.

because in the latter case the slack is often not large enough to make shutdown beneficial.

LAMPS+PS improves upon LAMPS mostly for coarse-grain tasks. Again, the main reason for this is that for fine-grain tasks, the slack is often not large enough to make shutting down worthwhile. With coarse grain tasks, however, a significant amount of energy can be saved by shutting processors down temporarily. The improvement of LAMPS+PS over LAMPS is typically less than the improvement of S&S+PS over S&S. This is because in LAMPS the static dissipation is already reduced by using a smaller number of processors compared to S&S. For coarse-grain tasks, the maximum improvements by

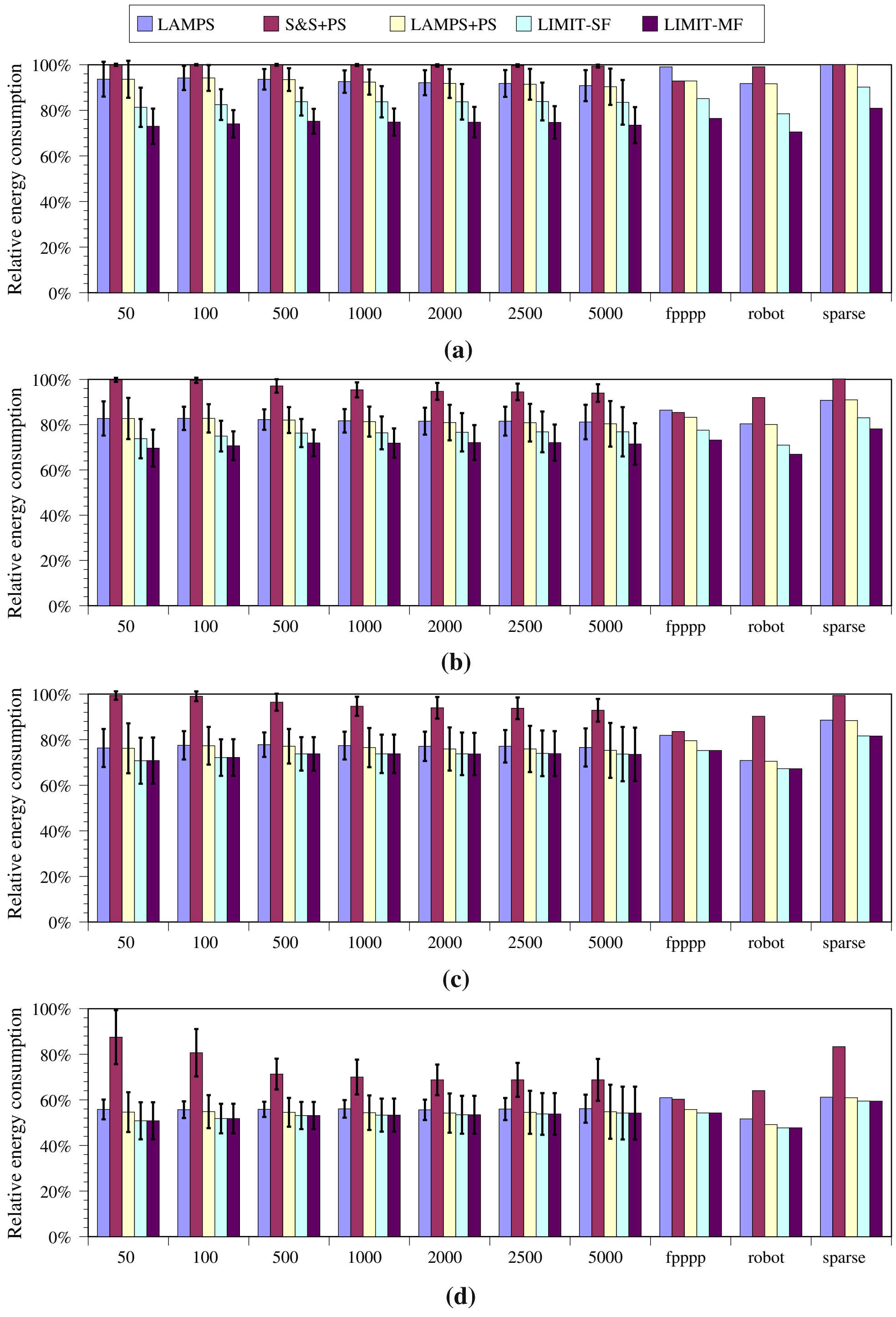


Figure 11 Energy consumption for fine-grain tasks. (**a**) Deadline = $1.5 \times CPL$. (**b**) Deadline = $2 \times CPL$. (**c**) Deadline = $4 \times CPL$. (**d**) Deadline = $8 \times CPL$.

LAMPS+PS upon LAMPS are 12% and 18%, for deadlines of 1.5× and 8× the CPL respectively.

For coarse-grain tasks, the total improvement by LAMPS+PS upon S&S is 16% on average, with a maximum of 46% for deadlines of 1.5× the CPL and a maximum of 73% for deadlines of 8× the CPL. For fine-grain tasks, LAMPS+PS improves upon S&S by 8% on average, with a maximum of 40% for deadlines of 1.5× the CPL and a maximum of 71% for deadlines of 8× the CPL.

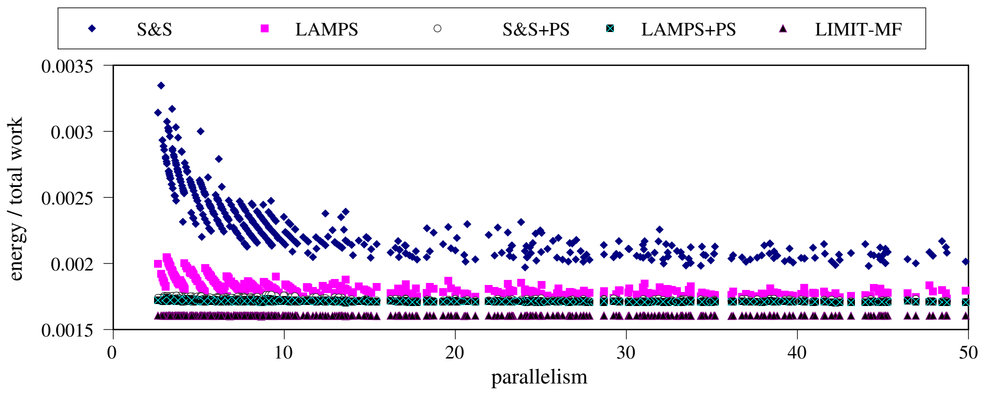


Figure 12 Energy/total work as a function of the average amount of parallelism for coarse-grain tasks. Each dot represents one task graph.

LIMIT-SF in Figs. 10 and 11 gives an upper limit on the energy savings using our current single-frequency model. Using S&S as the baseline and LIMIT-SF as the maximum, it shows that LAMPS+PS attains more than 94% of the possible energy reduction with coarse-grain tasks, for all combinations of benchmarks and deadlines. For fine-grain tasks and strict deadlines (1.5× the CPL), LAMPS+PS achieves more than 50% of the potential savings on 54% of the benchmarks. With less strict deadlines, LAMPS+PS attains more than 88% of the possible savings on all benchmarks.

In Figs. 10 and 11, Limit-MF is an indication for the possible improvements that could be attained by allowing the processors to run at a different frequency, and by allowing these frequencies to change over time. The results indicate that there is very little room for improvements when the deadline is relatively loose. For stricter deadlines, some savings may be attained, but mostly for fine-grain tasks. In the case of fine-grain tasks with strict deadlines, the periods of inactivity are often too small to make shutting down worthwhile. In this case, allowing varying frequencies might result in some additional savings. However, when the deadline is less strict and/or the task graph is fairly coarse-grained, shutting down processors becomes worthwhile. In this case, scheduling tasks at different frequencies will not provide a significant improvement.

To further explain why LAMPS and LAMPS+PS provide significant energy savings for certain task graphs, Figs. 12 and 13 depict the total energy divided by the total work as a function of the average amount of parallelism. Figure 12 depicts these results for coarse-grain tasks, while Fig. 13 shows the results for fine-grain tasks. In both cases, a deadline of 2 × the CPL is used. The total energy has been divided by the total work because there is almost a linear relationship between them. The average amount of parallelism is defined as the total work divided by the CPL. A linked list, for example, has an average amount of parallelism of 1. Each dot represents one task graph, and both figures depict the results for randomly generated graphs with 1000, 2000, 2500, and 3000 nodes.

From the figures it can be seen that the energy consumption per unit of work for S&S increases significantly when the average amount of parallelism becomes small. The same is visible for S&S+PS with

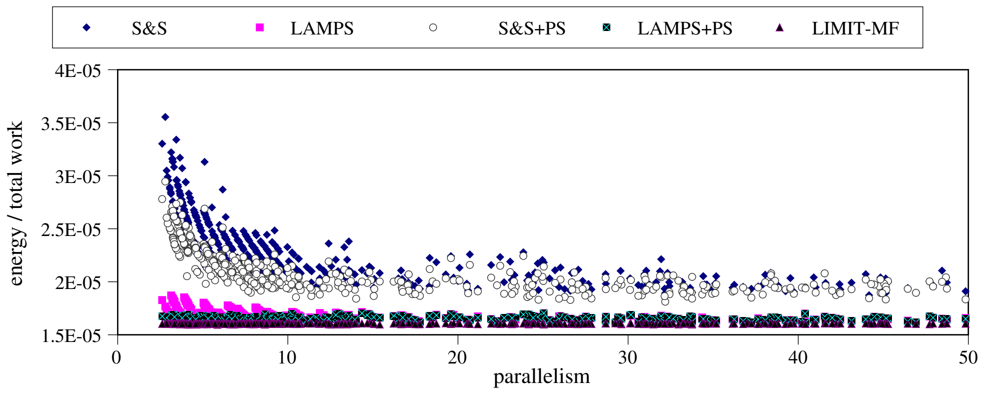


Figure 13 Energy/total work as a function of the average amount of parallelism for fine-grain tasks. Each dot represents one task graph.

Table 3 Energy consumption for the MPEG-1 benchmark using various approaches.

	S&S	LAMPS	S&S+PS	LAMPS+PS	LIMIT-SF	LIMIT-MF
Energy	18.116	13.290	10.949	10.947	10.940	10.940
# of processors	7	3	7	6	N/A	N/A

fine-grain tasks. This shows that especially when the average amount of parallelism is small, the energy consumption will increase when the option to shut down processors is not available or cannot be used effectively. The reason for this is that S&S will try to use as many processors as possible, but when the parallelism is low, they will be idle but continue to consume energy. For fine-grain tasks, the idle periods are often not long enough to save energy by shutting processors down temporarily, which is why S&S+PS with fine-grain tasks consumes significantly more energy than LAMPS and LAMPS+PS. For both LAMPS and LAMPS+PS, a small amount of parallelism has no significant effect on the energy consumption per unit of work, as both approaches can decide to use fewer processors.

The clustered results in these figures, especially visible when parallelism is low, are schedules that employ the same number of processors. For solutions with the same number of processors, the energy consumption decreases as the average parallelism more closely approaches the number of employed processors. Again, this effect is most clearly visible for S&S and S&S+PS, which employ as many processors as can effectively be used to exploit parallelism.

5.3 Results for MPEG-1

Figure 9 depicts the task graph for the MPEG-1 benchmark. The results from experiments with this benchmark are presented in Table 3. Similar to the previous experiments, these numbers were obtained by scheduling the task graph using the heuristics described in Section 4 and by measuring the energy consumption using the models described in Section 3.

When S&S is used to schedule this graph, it employs as many processors as can be used to reduce the makespan of the graph, which in this case is 7 processors. LAMPS, on the other hand, determines that using 3 processors is more efficient, and is hence able to reduce the energy consumption by more than 26% compared to S&S. S&S+PS also uses the maximum number of processors, but being able to shut processors down temporarily, it reduces the energy consumption by almost 40% compared to S&S. LAMPS+PS reduces the energy consumption by nearly the same amount as S&S+PS, albeit using 1 processor less. From this we can conclude that the periods of slack in the schedule are long enough to offset the cost of using one processor more. Furthermore, the results for S&S+PS and LAMPS+PS are extremely close to the lower limits LIMIT-SF and LIMIT-MF. From this we can conclude that it will not be possible to further reduce the energy consumption by using a different scheduling algorithm or by allowing processors to run at different and/or changing frequencies.

6 Conclusions and Future Work

As feature sizes keep decreasing, the contribution of leakage current to the total energy consumption is expected to increase. Depending on the amount of slack that remains before the deadline, the amount of parallelism, and the granularity of the application, voltage scaling as well as shutting down processors can be used to reduce the energy significantly. At the same time, it is important not to employ too many processors.

We have shown that our best approach, LAMPS+PS, reduces the energy consumption of a parallel MPEG-1 implementation by almost 40% compared to S&S. For a set of randomly generated task graphs, LAMPS+PS reduces the total energy consumption by up to 46% for tight deadlines and up to 73% for loose ones compared to the S&S algorithm. For coarse-grain tasks and a single frequency, LAMPS+PS attains more than 94% of the possible energy reduction, i.e., the energy reduction achieved by LIMIT-SF compared to S&S. Since LIMIT-SF is independent of the scheduling algorithm, this implies that there is almost no room left for improvement by using other scheduling algorithms than EDF.

Employing too many processors can significantly increase the total energy consumption, especially if the average amount of parallelism is low. When the amount of parallelism and the number of employed processors are higher, using additional processors is relatively less costly.

Even when multiple frequencies are allowed, LAMPS+PS reduces the energy consumption close to the theoretical limit (LIMIT-MF). For loose deadlines ($4\times$ or $8\times$ the CPL), LIMIT-MF consumes the same amount of energy as LIMIT-SF, and so LAMPS+PS again attains over 94% of the potential savings with coarse-grain tasks. As a result, it will be nearly impossible to reduce the overall energy consumption further by using other scheduling algorithms that produce schedules in which different processors can run at different frequencies and in which the frequency can change over time. Applications consisting of relatively fine-grain tasks, on the other hand, might benefit from using other scheduling approaches. However, since LIMIT-MF does not take the deadline into account, real scheduling approaches will probably not reach this limit. Consequently, the actual benefit from having multiple frequencies will probably be much less. We have shown that having processors run at their own frequency, as well as having this frequency change over time, does not provide a significant improvement, especially with coarse-grain task graphs and a relaxed deadline. For more fine-grain tasks and stricter deadlines, some improvements might be attained by using multiple frequencies or by using other scheduling algorithms such as the algorithm that maximizes the slack proposed in [1] or the integrated approach described in [18]. We intend to investigate the impact of these techniques on fine-grain task graphs in more detail in future research.

References

1. Zhu, D., Melhem, R., & Childers, B. (2003). 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.
2. de Langen, P. J., & Juurlink, B. (2006). Leakage-aware multiprocessor scheduling for low power. In *Proc. Int. Parallel and Distributed Processing Symp.*
3. de Langen, P. J., & Juurlink, B. (2007). Trade-offs between voltage scaling and processor shutdown for low-energy embedded multiprocessors. In *Proc. Int. Workshop on Computer Systems: Architectures, Modelling, and Simulation* (pp. 75–85).
4. ARM Ltd (2008). ARM11 MPCore. http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html.
5. Hofstee, H. (2005). Power efficient processor architecture and the cell processor. In *Proc. Int. Symp. on High-Performance Computer Architecture* (pp. 258–262).
6. Borkar, S. (1999). Design challenges of technology scaling. *IEEE Micro, 19*(4), 23–29.
7. Duarte, D., Vijaykrishnan, N., Irwin, M., & Tsai, Y. (2002). Impact of technology scaling and packaging on dynamic voltage scaling techniques. In *Proc. IEEE Int. ASIC/SOC Conf.*
8. Burd, T. D., & Brodersen, R. W. (1995). Energy efficient CMOS microprocessor design. In *Proc. Hawaii Int. Conf. on System Sciences* (pp. 288–297).
9. Pering, T., & Broderson, R. (1998). Dynamic voltage scaling and the design of a low-power microprocessor system. In *Proceedings of the Power Driven Microarchitecture Workshop, attached to ISCA98.*
10. Intel Corp. Intel XScale Technology. http://www.intel.com/design/intelxscale/.
11. Jha, N. (2005). Low-power system scheduling, synthesis and displays. *IEE Proceedings on Computers and Digital Techniques, 152*(3), 344–352.
12. Gruian, F., & Kuchcinski, K. (2001). LEneS: Task scheduling for low-energy systems using variable supply voltage processors. In *Proc. Conf. on Asia South Pacific Design Automation* (pp. 449–455).
13. Jejurikar, R., Pereira, C., & Gupta, R. (2004). Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proc. Conf. on Design Automation* (pp. 275–280).
14. Quan, G., Niu, L., Hu, X. S., & Mochocki, B. (2004). Fixed priority scheduling for reducing overall energy on variable voltage processors. In *Proc. Int. Real-Time System Symposium* (pp. 309–318).
15. Lee, Y., Reddy, K., & Krishna, C. (2003). Scheduling techniques for reducing leakage power in hard real-time systems. In *Proc. Euromicro Conf. on Real-Time Systems* (pp. 105–112).
16. Irani, S., Shukla, S., & Gupta, R. (2003). Algorithms for power savings. In *Proc. ACM-SIAM Symp. on Discrete Algorithms* (pp. 37–46).
17. Zhang, Y., Hu, X. S., & Chen, D. Z. (2002). Task scheduling and voltage selection for energy minimization. In *Proc. Conf. on Design Automation* (pp. 183–188).
18. Kianzad, V., Bhattacharyya, S. S., & Qu, G. (2005). CASPER: An integrated energy-driven approach for task graph scheduling on distributed embedded systems. In *Proc. IEEE Int. Conf. on Application-Specific Systems, Architecture Processors* (pp. 191–197).
19. Varatkar, G., & Marculescu, R. (2003). Communication-aware task scheduling and voltage selection for total systems energy minimization. In *Proc. Int. Conf. on Computer-Aided Design* (pp.510–517).
20. Gonzalez, R., Gordon, B., & Horowitz, M. (1997). Supply and threshold voltage scaling for low power CMOS. *IEEE Journal of Solid-State Circuits, 32*(8), 1210–1216.
21. Martin, S., Flautner, K., Mudge, T., & Blaauw, D. (2002). Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proc. Int. Conf. on Computer-Aided Design* (pp. 721–725).
22. Andrei, A., Schmitz, M., Eles, P., Peng, Z., & Al-Hashimi, B. M. (2004). Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. In *Proc. Conf. on Design, Automation and Test in Europe* (pp. 518–525).
23. Yan, L., Luo, J., & Jha, N. K. (2003). Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. In *Proc. Int. Conf. on Computer-Aided Design* (pp. 30–37).
24. Xu, R., Zhu, D., Rusu, C., Melhem, R., & Mossé, D. (2005). Energy-efficient policies for embedded clusters. In *Proc. ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems* (pp. 1–10).

25. Liberato, F., Lauzac, S., Melhem, R., & Mossé, D. (1999). Fault tolerant real-time global scheduling on multiprocessors. In *Proc. Euromicro Conf. on Real-Time Systems* (pp. 252–259).
26. Kahn, G. (1974). The semantics of a simple language for parallel programming. *Information Processing, 74*, 471–475.
27. Kasahara, H., Tobita, T., Matsuzawa, T., & Sakaida, S. (2008). Standard task graph set. http://www.kasahara.elec.waseda.ac.jp/schedule/.

Pepijn de Langen was born in Groningen, the Netherlands, in 1976. He received the M.Sc. degree in electrical engineering from Delft University of Technology in 2003. He is currently at the final stage of his Ph.D. in the Computer Engineering Laboratory at Delft University of Technology. His research interests include advanced computer architectures, memory systems, hardware/software co-design, and techniques for reducing power consumption.

Ben Juurlink is an associate professor in the Computer Engineering Laboratory of the Faculty of Electrical Engineering, Mathematics, and Computer Science at Delft University of Technology, the Netherlands. He received the MSc degree in computer science, from Utrecht University, Utrecht, the Netherlands, in 1992, and the Ph.D. degree also in computer science from Leiden University, Leiden, the Netherlands, in 1997. His research interests include instruction-level parallel processors, application-specific ISA extensions, low power techniques, and hierarchical memory systems. He has (co-)authored more than 50 papers in international conferences and journals and is a senior member of the IEEE, a member of the ACM, and a member of the HiPEAC Network of Excellence.