A Scheduling Selection Process for Energy-efficient Task Execution on DVFS Processors

Thomas Rauber
Computer Science Department,
Universität Bayreuth,
95440 Bayreuth, Germany
rauber[at]uni-bayreuth.de

Gudula Rünger
Computer Science Department,
Technische Universität Chemnitz,
09107 Chemnitz, Germany
ruenger[at]cs.tu-chemnitz.de

To cite this version:
Thomas Rauber, Gudula Rünger: A scheduling selection process for energy-efficient task execution on DVFS processors. Concurr. Comput. Pract. Exp. 31(19) (2019), ISBN 1532-0626, DOI: 10.1002/cpe.5043 John Wiley & Sons

Abstract

The efficient execution of parallel programs with respect to execution time and energy consumption is a major concern and often scheduling methods are used to achieve a good performance. In this article, we consider the problem of scheduling a set of independent tasks on a parallel system with homogeneous execution units providing frequency scaling. The set of tasks has the property that the tasks exhibit a task-specific inhomogeneous and non-linear behavior of their specific time-energy relation. Also, it is assumed that the execution time as well as the energy consumption behave in a non-linear manner with respect to frequency scaling. For the assignment of these tasks to execution units, we propose a scheduling selection process combining scheduling algorithms, which determine a task assignment, with a subsequent selection of frequency scaling, which we call schedule execution modes. This process builds a rich set of alternative schedule execution modes from which efficient (or Pareto-optimal) schedule execution modes can be selected. Experiments are done for the SPEC CPU benchmarks. Experimental results illustrate that the enriched scheduling process leads to task assignments resulting in an efficient execution on DVFS processors.

1 Introduction

The scheduling of tasks on a set of resources is an important area of parallel computing, and many algorithms have been proposed in the past, using different objective functions, such as makespan, maximum lateness, total weighted completion time, or total weighted tardiness, see [13] for a detailed overview. The tasks to be scheduled may be independent from each other or may have different types of dependencies, for example represented by a directed acyclic graph. Most scheduling algorithms considered in the past have focused on the execution time of the tasks and objective functions derived from execution times. However, in recent years the energy consumption of applications is considered to be of growing importance, and hardware manufacturers have developed a variety of power-aware system features, including multicore-on-a-chip processors, core-independent functional units, dynamic voltage and frequency scaling (DVFS), or clock gating to support energy-efficient computing [26, 4, 6].

The modified focus of interest towards energy aspects has led to a re-investigation of scheduling algorithms and the consideration of the resulting energy consumption as objective function [1, 33]. In this article, we follow this research direction and investigate the impact of task scheduling algorithms on the performance and the energy consumption for real applications. When considering the execution of tasks from real-world applications on recent DVFS processors, it can be observed that the energy and the time consumed for the computations exhibit a complex behavior with a potentially non-linear reaction to frequency scaling due to variations in the power consumption. Furthermore, each task may react in a different individual way. The non-linear and in-homogeneous behavior has to be taken into account when determining an assignment of tasks to DVFS processors. The efficient execution with respect to energy and time may involve conflicting objectives leading to a multicriteria decision problem [3] for choosing an optimal combination of task schedule and scaling factors of the operational frequency.

In this article, we consider the problem of scheduling a set of independent tasks on a homogeneous set of identical DVFS processors. As objective function, the makespan or the energy consumption is taken, respectively. There is no assumption about the behavior of the execution time and the energy consumption with respect to frequency scaling, meaning that an in-homogeneous and non-linear behavior can be handled. We propose a schedule selection process for finding an efficient mapping of a set of tasks representing real-world applications to DVFS processors. In particular, we investigate scheduling algorithms using the makespan as well as the energy consumption as objective function. Of special interest is the question, whether the schedules derived by an energy-oriented scheduling really lead to a smaller overall energy consumption or whether time-oriented scheduling can also lead to a good energy usage. The choice of an optimal frequency of all or of individual execution units is intertwined with the determination of schedules resulting in a multitude of possible execution alternatives of a given set of tasks.

The scheduling selection process consists of several steps. First, the proposed scheduling algorithms are applied with different objective functions, i.e., energy or time, and with different operational frequencies. This yields a candidate set of alternative schedules, each providing a task assignment and a frequency setting. The schedules obtained for a specific frequency are applied to evaluate the makespan and the energy consumption for the other frequencies. Thus, each of the schedule candidates is executed for several execution specifications of the operational frequency, which is called schedule execution modes to express that the schedule is executed in a special mode of the operational frequency. The resulting set of schedule execution modes is further enriched by frequency scaling of individual execution units which may potentially reduce the energy consumption without increasing the execution time. In total, the scheduling process provides a rich set of alternatives each consisting of an assignment of tasks to processors and a frequency setting for each processor. To find efficient schedules, the set of alternatives is evaluated with respect to the objective function. A set of non-dominated alternatives, called set of efficient alternatives, with respect to the multicriteria objective involving execution time and energy consumption is determined. A last step is to choose a final solution from the set of efficient alternatives due to the individual needs of an execution run, e.g. minimum execution time or minimum energy consumption or a combination of both.

To test the scheduling algorithms and the frequency scaling, we use tasks that have been obtained from executing the SPEC CPU benchmarks, see `www.spec.org`, on an Intel Core i7 Haswell processor. The experimental results exhibits the inhomogeneous and non-linear behavior of real-world applications with respect to execution time and energy consumption on DVFS processors.

The contribution of the article is an in-depth investigation of the schedules resulting from time-oriented and from energy-oriented scheduling algorithm along with an analysis of the benefits resulting from individual DVFS settings for the different processors. Also, a detailed comparison of the makespan and the energy consumption of schedules is provided. The investigation reveals that a time-oriented scheduling may lead to even better results than an energy-oriented scheduling for the makespan and also for the energy consumption, if the most appropriate operational frequencies and schedules are used for DVFS. The scheduling selection process proposed takes this into account and accompanying theoretical results explain the effects.

The article is structured in the following way: Section 2 considers the power and energy consumption of tasks when using DVFS. Section 3 presents the scheduling algorithms considered. Section 4 describes the multicriteria decision problem and the scheduling selection process proposed. Section 5 investigates the resulting makespans and energy consumptions of schedules for SPEC benchmark tasks and the dependence on the frequency and the number of execution units. Section 6 discusses related work and Section 7 concludes.

2 Power and energy behavior of application programs

2.1 Power and energy models

The energy E consumed for the execution of an application code depends on the execution time T on the given hardware platform and the power drawing P during the execution. P may vary during the execution of the code and may also depend on the specific execution behavior of the application considered. The application-specific

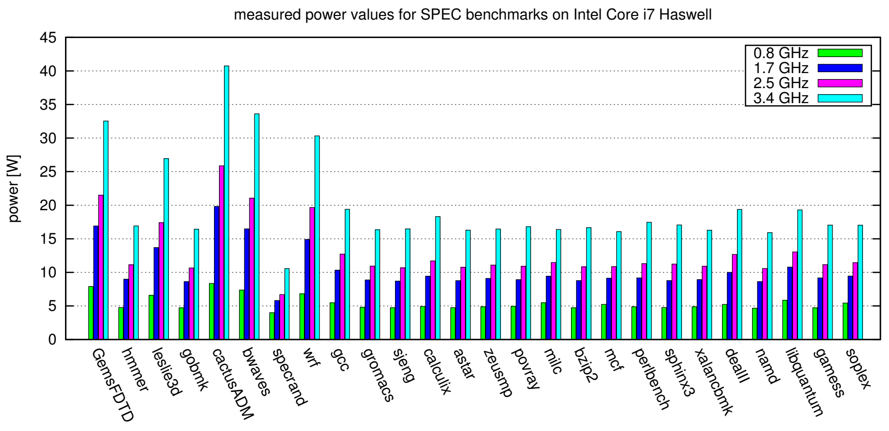


Figure 1: Power consumption of the SPEC CPU benchmarks for four selected frequencies.

behavior may come from different sources, such as varying usages of internal functional units or caches, or memory accesses. The resulting energy E is expressed as $E = \int_{t=0}^{t_{end}} P(t)dt$, assuming that the program is executed from time $t = 0$ to $t = t_{end}$. The power P is measured in Watt (W), the time is measured in seconds (s), and the energy is measured in Joule (J).

In the following, we consider DVFS systems with p_{max} cores and operational frequencies f ranging between a minimum frequency f_{min} and a maximum frequency f_{max}. In the following, we only deal with frequency scaling, not voltage scaling. The frequency can be changed only in discrete steps. Frequency scaling for DVFS processors can be expressed by a dimensionless scaling factor $s \geq 1$ which describes a smaller frequency $f \leq f_{max}$ in relation to the maximum possible frequency f_{max} as $f = f_{max}/s$. The power drawing $P(t)$ varies with the operational frequency f chosen, which is expressed by the power being a function of s, i.e., $P = P(s,t)$. Analogously, the energy consumption also depends on s, which is expressed by an additional parameter, i.e., $E(s) = \int_{t=0}^{t_{end}(s)} P(s,t)dt$.

Power models for DVFS processors often distinguish the dynamic power consumption P_{dyn}, which is related to the computational activity of the processor, and the static power consumption P_{stat}, which is intended to capture the leakage power consumption occurring also during idle times [20]. P_{dyn} depends on the frequency f and, thus, the scaling factor s, and P_{stat} is often treated to be independent from s. The dependence of P_{dyn} on f is often assumed to have the form $P_{dyn}(f) = \gamma \cdot f^3$ with a suitable constant γ, or when using the corresponding scaling factor s as $P_{dyn}(s) = s^3 \cdot P_{dyn}(1)$ where $P_{dyn}(1)$ is the dynamic power consumption in the un-scaled case. The total power consumption is the sum of both parts, i.e., $P(s,t) = P_{dyn}(s,t) + P_{stat}$. For simplicity, it is often assumed that $P(s,t)$ is constant during the execution of a specific application program, leading to: $E(s) = P(s) \cdot T(s)$, where the overall execution time $T(s) = t_{end}(s)$ of the application program considered depends on the scaling factor s. In practice, E is a discrete function in s, since the frequency cannot be changed continuously.

As an illustration of the power consumption behavior of applications, Fig. 1 shows the power consumption of the SPEC CPU 2006 benchmarks for different frequencies (0.8 GHz, 1.7 GHz, 2.5 GHz, 3.4 GHz) on an Intel Core i7 Haswell processor. The figure shows that the power consumed for executing the individual benchmark programs varies significantly among the benchmarks, reflecting the fact that the power consumed is strongly application-specific. Also, the power consumption is different for different operational frequencies and increases with the frequency. Moreover, different benchmark programs exhibit different power increase factors.

2.2 Energy consumption of task-based applications

In this article, we consider a task-based programming model in which the parallel program to be executed consists of a set $\mathcal{T}$ of n independent tasks without ready times. Each task can be executed sequentially by any of the processors or cores provided by the parallel execution platform. A homogeneous parallel platform with p_{max} identical processing units (processors or cores) supporting DVFS is assumed. Several tasks can be executed in parallel to each other, each one on a separate processor or core, and a processor can fetch the next task for execution

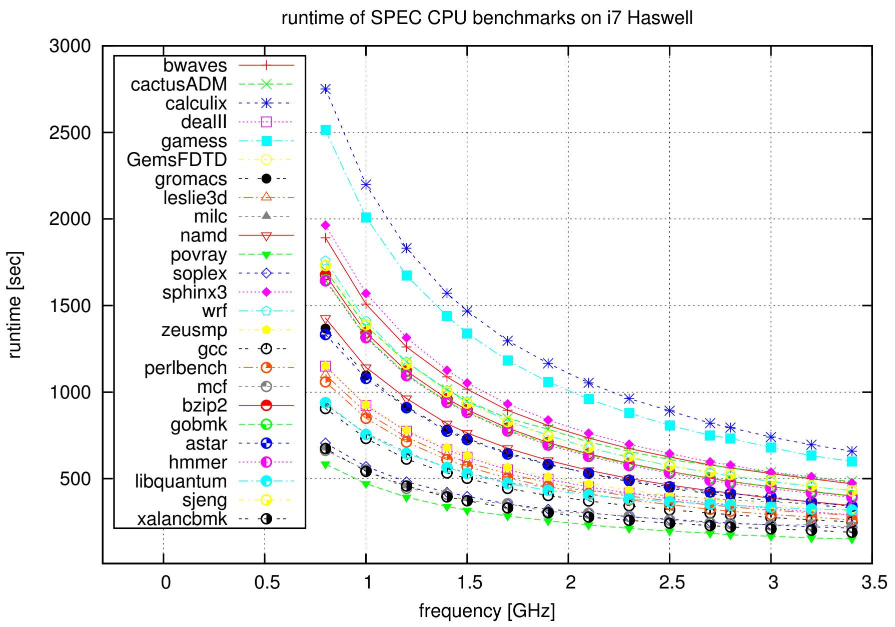


Figure 2: Runtime behavior of the SPEC CPU benchmarks for varying frequencies.

as soon as it becomes idle. The assignment of the tasks to processors is done according to a schedule resulting from a scheduling algorithm for sequential tasks; appropriate algorithms are proposed in Sect. 3. In the following, we introduce the terminology and definitions used. The **assignment function** defining the mapping of tasks to processing unit is denoted as:

$$A : \mathcal{T} \to \{1, \ldots, p_{max}\}. \tag{1}$$

The overall execution time of a task-based program is determined by the execution times of the individual tasks $x \in \mathcal{T}$ and the coordination and idle times resulting from a specific schedule. Since we consider independent tasks executed without delay, idle times can occur only after the last task on a processor has been finished. The execution time of a task $x \in \mathcal{T}$ for a frequency scaling factor s is denoted by $T_x(s)$. The execution time of a task can be given in different ways, which can be a measured execution time on a specific hardware platform, a predicted time using an analytical prediction technique, such as proposed in [12, 11], or relative values between the execution times of the tasks. In this article, we use measured execution times. Experiments show that the execution times of the tasks do not depend linearly on the operational frequency or the corresponding scaling factor. Instead, an application-specific non-linear behavior can be observed, which is illustrated in Fig. 2 giving the measured execution times of the SPEC CPU benchmarks on an Intel Core i7 Haswell processor for different operational frequencies. The runtimes reported in Fig. 2 have been obtained by measuring the execution times of the different benchmarks in isolation fixing the operational frequency with the `cpu-freqset` tool.

For a specific assignment function A, the **accumulated execution time** $T^A_{acc}(p, s)$ for a processor $p \in \{1, \ldots, p_{max}\}$ is calculated by summing up all execution times $T_x(s)$ of the tasks x assigned to processor p, i.e. for which $A(x) = p$, and using scaling factor s for execution. This results in:

$$T^A_{acc}(p, s) = \sum_{A(x)=p} T_x(s). \tag{2}$$

The **overall execution time** $T^A_{total}(s)$ of the entire task-based parallel program is determined by the processor having the longest execution time, which is calculated as the maximum of $T^A_{acc}(p, s)$ over all p according to:

$$T^A_{total}(s) = \max_{p=1,\ldots,p_{max}} T^A_{acc}(p, s) \tag{3}$$

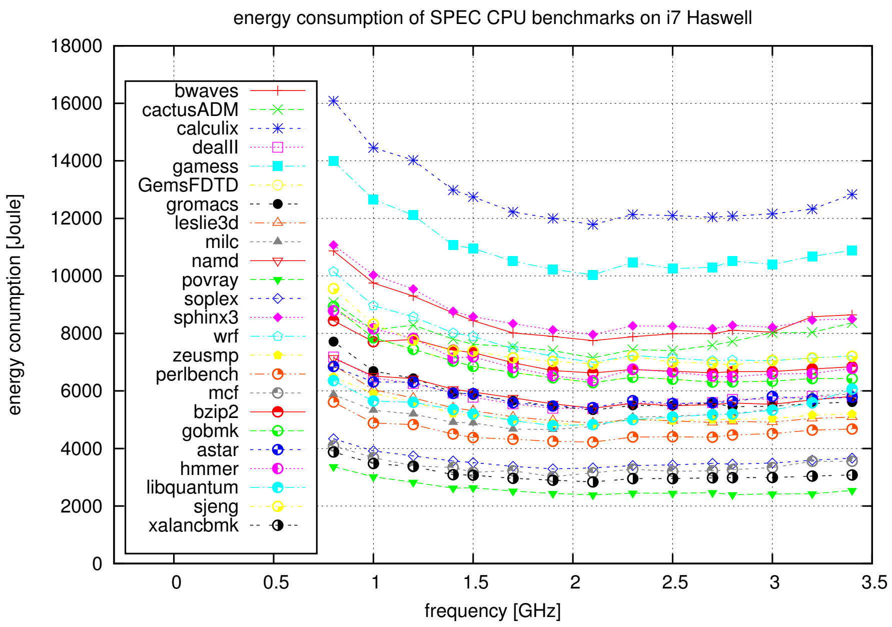


Figure 3: Behavior of the energy consumption of the SPEC CPU benchmarks for varying frequencies.

In the context of scheduling, $T^A_{total}(s)$ is often called the makespan of the assignment function A.

Analogously to the accumulated execution time, we define an accumulated energy consumption for processor p, denoted as $E^A_{acc}(p, s)$, which is calculated from the energy values $E_x(s)$ consumed by the tasks x using frequency scaling factor s. Similarly to the execution time, the energy consumption of a task typically depends in a non-linear way on the operational frequency, which is illustrated in Fig. 3. For the energy consumption, the non-linear and application-specific dependence of the power consumption on the frequency also plays an important role, see Fig. 1. The **accumulated energy consumption** $E^A_{acc}(p, s)$ of a processor $p \in \{1, \ldots, p_{max}\}$ for a specific assignment function A is given by:

$$E^A_{acc}(p, s) = \sum_{A(x)=p} E_x(s). \tag{4}$$

The overall energy consumption of the entire task-based program is denoted as $E^A_{total}(s)$ and is calculated by summing up the energy consumption of all processors participating in the execution of the program. Since the execution time is determined by the processor with the longest execution time defining the makespan, the makespan has to be taken into account for all other processors even if the active execution time is shorter. The time between the end of the active execution time of a processor p and the makespan defines the **idle time of the processor** p, denoted as

$$T^A_{idle}(p) = (T^A_{total}(s) - T^A_{acc}(p, s)), \tag{5}$$

during which no calculations are contributed by p, but nevertheless static power is consumed. This assumes that a processor is not switched off after it has executed its last task but remains active and consumes static power. An alternative would be to switch off processors or cores during idle times. This could avoid the consumption of static power, but requires time for a change of the processor power management state. In the following, we assume that the processors are not switched off but remain active during idle times. The assumption can be justified by the fact that the static power consumption is typically much smaller than the dynamic power consumption and that all processors should be available as soon as possible for executing a following schedule. Thus, for calculating the overall energy consumption it has to be taken into account that processors might have idle times during which they consume static power. This results in the following formula for the **overall energy consumption**

$$E^A_{total}(s) = \sum_{p\in\{1,\ldots,p_{max}\}} \left(E^A_{acc}(p, s) + (T^A_{total}(s) - T^A_{acc}(p, s)) \cdot P_{stat}\right). \tag{6}$$

The formulas introduced here, i.e. the accumulated and the total execution time and energy consumption, are used as cost measures in the scheduling algorithms presented in Sect.3.

3 Time-oriented and Energy-oriented Scheduling algorithms

3.1 Scheduling problem

We consider a set $\mathcal{T}$ of $n = \mid \mathcal{T} \mid$ independent sequential tasks without ready times, denoted as $\mathcal{T} = \{x_1, \ldots, x_n\}$. The execution platform has p_{max} homogeneous processors or cores which can be independently set to an operational frequency $f \in \mathcal{F}$ where $\mathcal{F}$ is a finite set of discrete frequencies, which lie between a minimum frequency f_{min} and a maximum frequency f_{max}. Each frequency $f \in \mathcal{F}$ has an associated scaling factor $s = f_{max}/f$. The size of $\mathcal{F}$ is denoted as n_F. Each task $x \in \mathcal{T}$ can be executed with any frequency $f \in \mathcal{F}$ resulting in execution time $T_x(s)$ and energy consumption $E_x(s)$ where s is the corresponding scaling factor to f. We assume that the frequency cannot be changed during the execution of x.

To be consistent with the measurements on real DVFS processors, see Figs. 2 and 3, no assumptions are made about a functional relation between the operational frequency or scaling factor, respectively, and the resulting execution time or energy consumption of a task. The only assumption that is made is a monotonicity behavior of the execution time, i.e.:

$$T_x(s_1) \leq T_x(s_2) \text{ for } s_1 < s_2,$$

which means that an increase of the operational frequency leads to a reduction of the execution time, see Fig. 2. Such a monotonicity behavior does not exist for the energy consumption, see Fig. 3, since the reduction of the execution time is accompanied by an increase of the power consumption. Considering the povray application as specific example, the execution time is reduced from 584.9 sec to 151.5 sec when increasing the operational frequency from 0.8 GHz to 3.4 GHz, while the overall power consumption increases from 5.75 Watt to 16.76 Watt. As a result, the energy consumption decreases from 3365 Joule to 2539 Joule. For most of the applications, the energy consumption is quite large for small frequencies and decreases until about 2 GHz. Then the energy consumption rises again until the largest frequency available is reached. In total, this leads to an u-shape diagram for the energy consumption of each application as observed in Fig. 3.

For the assignment of tasks to processors, we use a two-step approach. In the first step, we use the same fixed frequency f with scaling factor s for all tasks and all processors and determine an assignment A with the goal to either minimize the makespan

$$\text{minimize } T^A_{total}(s) \text{ with } T^A_{total}(s) \text{ given according to Equ. (3)}$$

or the overall energy consumption

$$\text{minimize } E^A_{total}(s) \text{ with } E^A_{total}(s) \text{ given according to Equ. (6).}$$

This scheduling step is described in Sections 3.2 and 3.3. In a second step, the frequencies of the processors are modified and adjusted to reduce the overall energy consumption further. The second step will be addressed in Section 4.

3.2 Time-oriented scheduling

For the time-oriented scheduling algorithm, the execution time (or makespan) is chosen as objective function. At the beginning, the tasks $x \in \mathcal{T}$ are sorted in descending order of their execution times. Since the operational frequency $f \in \mathcal{F}$ has a considerable impact on the execution time, the sorting is done separately for the execution times measured for each available frequency. For a selected frequency f and its corresponding scaling factor s, the scheduling algorithm assigns the tasks from the task list one after another to a newly selected processor in a stepwise way. In step j, $j = 1, \ldots, n$, the newly selected processor is determined with respect to the partial accumulated execution times $T_{[1:j-1]}(p, s)$ which have been computed for all p after the previous step $j - 1$. The

Algorithm 1 Time-oriented scheduling algorithm to generate a T-schedule.

```
1:  procedure T-SCHEDULE                    ▷ using execution time as objective function
2:      Select frequency $f \in \mathcal{F}$ with scaling factor $s$;
3:      Sort tasks $\{x_1, \ldots, x_n\}$ such that $T_{x_1}(s) \geq T_{x_2}(s) \geq \ldots \geq T_{x_n}(s)$;
4:      for ($j = 1, \ldots, n$) do
5:          Assign $x_j$ to processor $p$ with $T_{[1:j-1]}(p, s)$ minimal, see Equ. (7);
6:      end for
7:      denote the resulting assignment function as $A$;
8:      Compute makespan $T^A_{total}(s) = \max_{1 \leq p \leq p_{max}} T_{[1:n]}(p, s)$, see Equ. (3);
9:      Compute total energy $E^A_{total}(s)$ according to Equ. (6);
10: end procedure
```

partial accumulated execution time for processor p sums up the execution times $T_{x_k}(s)$ of all tasks x_k, $1 \leq k \leq j$, $j \leq n$ assigned to p so far, where n is the size of the task set $\mathcal{T}$. The **partial accumulated execution time** is defined as:

$$T_{[1:j-1]}(p, s) = \sum_{\substack{1 \leq k < j \\ A(x_k) = p}} T_{x_k}(s) \,. \tag{7}$$

The processor p selected in step j is the processor with the smallest partial accumulated execution time according to Equ. (7). If more than one processor has this property, an arbitrary one of these processors is selected. The assignment step for the first task is covered, since $T_{[1:0]}(p, s) = 0$ for all processors. The makespan of the final schedule is determined by the processor with the highest accumulated execution time $T_{[1:n]}(p, s)$ after step n. The corresponding energy consumption for the resulting schedule has to take the idle time of all other processors into account and is computed according to Equ. (6). The pseudocode of the scheduling algorithm is given in Algorithm 1. For a given task set and the same number of processors, the scheduling algorithm may lead to different schedules when selecting different frequencies $f \in \mathcal{F}$.

Algorithm 1 determines schedules under the assumption that all processors use the same operational frequency. The processor for which the largest execution time results determines the makespan for the specific frequency selected. The energy consumption may be reduced by adjusting the frequencies of the other processors in such a way that their idle times are reduced but their execution times are still below the makespan. Since the processor with the largest execution time is not scaled down, the makespan does not increase. The adaption of the frequencies of all processors except the processor determining the makespan is described in Section 4.2.

3.3 Energy-oriented scheduling

The energy-oriented scheduling given in Algorithm 2 works in a similar manner as Algorithm 1, but instead of using the execution times, it uses the energy values for sorting and assigning tasks to processors. For the assignment, the **partial accumulated energy** $E_{[1:j-1]}(p, s)$, $j = 1, \ldots, n$ for processor p is used:

$$E_{[1:j-1]}(p, s) = \sum_{\substack{1 \leq k < j \\ A(x_k) = p}} E_{x_k}(s) \,. \tag{8}$$

The computation of the makespan and the total energy of Algorithm 2 work analogously to Algorithm 1. After the last task has been assigned, the total energy consumption $E^A_{total}(s)$ is obtained according to Equ. (6). The execution time of each processor is now calculated by adding the execution times of all tasks assigned to this processor according to the energy-oriented schedule. The maximum of these execution times is the makespan given in Equ. (3). The schedules resulting from Algorithm 2 do not only differ for different frequencies, but can also differ from the schedules resulting from a time-oriented scheduling for the same frequency, which is illustrated by the experimental evaluation in Section 5. For the energy-oriented scheduling, the frequency adaptation for a schedule can be especially beneficial, since typically larger variations of the execution times of the individual processors result from Algorithm 2, giving a larger potential for frequency adjustments. This will be done in Section 4.

Algorithm 2 Energy-oriented scheduling algorithm to generate an E-schedule.

```
1:  procedure E-SCHEDULE                    ▷ using energy consumption as objective function
2:      Select frequency $f \in \mathcal{F}$ with scaling factor $s$;
3:      Sort tasks $\{x_1, \dots, x_n\}$ such that $E_{x_1}(s) \geq E_{x_2}(s) \geq \dots \geq E_{x_n}(s)$;
4:      for ($j = 1, \dots, n$) do
5:          Assign $x_j$ to processor $p$ with $E_{[1:j-1]}(p, s)$ minimal, see Equ. (8);
6:      end for
7:      denote the resulting assignment function as $A$;
8:      Compute makespan $T^A_{total}(s) = \max_{1 \leq p \leq p_{max}} T_{[1:n]}(p, s)$, see Equ. (3);
9:      Compute total energy $E^A_{total}(s)$ according to Equ. (6);
10: end procedure
```

Although the resulting makespan and overall energy consumption can be very different, the following observation concerning the monotonicity can be shown when comparing two different assignment functions A_1 and A_2 for a task set $\mathcal{T}$:

Lemma 1 *If $T^{A_1}_{total}(s) < T^{A_2}_{total}(s)$ for a given number of identical processors and a fixed scaling factor s, then $E^{A_1}_{total}(s) < E^{A_2}_{total}(s)$.*

Proof 1 *For a fixed scaling factor s, we have:*

$$E^{A_1}_{total}(s) = \sum_p \left(E^{A_1}_{acc}(p, s) + \left(T^{A_1}_{total}(s) - T^{A_1}_{acc}(p, s) \right) \cdot P_{stat} \right)$$

according to Equ. (6). Moreover the equality

$$\sum_p E^{A_1}_{acc}(p, s) = \sum_{x \in T} E_x(s) = \sum_p E^{A_2}_{acc}(p, s),$$

holds, which expresses that the sum of the energy consumptions of the active computation time of all processors $p \in \{1, \dots, p_{max}\}$ is identical for A_1 and for A_2, since the same tasks are used and all processors are identical. Similarly, summing up the task execution times of all processors is identical for A_1 and A_2

$$\sum_p T^{A_1}_{acc}(p, s) = \sum_{x \in T} T_x(s) = \sum_p T^{A_2}_{acc}(p, s).$$

From the two equations and the assumption $T^{A_1}_{total}(s) < T^{A_2}_{total}(s)$, the following estimation can be derived:

$$\begin{aligned} E^{A_1}_{total}(s) &= \textstyle\sum_p E_x(s) + p_{max} \cdot T^{A_1}_{total}(s) \cdot P_{stat} - \sum_p T^{A_1}_{acc}(p, s) \cdot P_{stat} \\ &= \textstyle\sum_p E^{A_2}_{acc}(p, s) + p_{max} \cdot T^{A_1}_{total}(s) \cdot P_{stat} - \sum T^{A_2}_{acc}(p, s) \cdot P_{stat} \\ &< \textstyle\sum_p E^{A_2}_{acc}(p, s) + p_{max} \cdot T^{A_2}_{total}(s) \cdot P_{stat} - \sum T^{A_2}_{acc}(p, s) \cdot P_{stat} \\ &= E^{A_2}_{total}(s) \end{aligned}$$

Thus, an assignment function with a smaller makespan also leads to a smaller energy consumption for a fixed scaling factor. This result expresses that the larger idle times for schedules with a larger makespan induces more static energy consumption for the idle times and therefore a larger overall energy consumption. However, this does not take frequency adaptation into account, which could be employed to fill the idle times with active computations. Moreover, the following lower bounds for the total execution time and the total energy consumption can be shown:

Lemma 2 *For a fixed scaling factor s, the value*

$$T_{low}(s) = \sum_{x \in \mathcal{T}} T_x(s) / p_{max} \tag{9}$$

Algorithm 3 Frequency adaptation for a given T-schedule or E-schedule.

```
1: procedure FrequencyAdaptation                ▷ adaptation of frequency of individual processors
2:     Let $A$ be a given T-schedule or E-schedule generated by Algorithm 1 or 2 for scaling factor $s$;
3:     Let $T^A_{total}(s)$ and $E^A_{total}(s)$ be the makespan and energy consumption of $A$;
4:     Let $p_e$ be the makespan processor, i.e., $T_{[1:n]}(p_e, s) = T^A_{total}(s)$;
5:     for (each $p \neq p_e$) do
6:         Select frequency for $p$ with $(T^A_{total}(s) - T^A_{acc}(p, s))$ minimal;
7:     end for
8: end procedure
```

is a lower bound for the achievable makespan and

$$E_{low}(s) = \sum_{x \in \mathcal{T}} E_x(s) \tag{10}$$

is a lower bound for the total energy consumption.

Proof 2 *For a fixed scaling factor s, the total amount of sequential execution time is $\sum_{x \in \mathcal{T}} T_x(s)$, which can be distributed to p_{max} processors. Thus, $T_{low}(s)$ is the lower bound for the makespan given in Equ. (3), which can be achieved if the tasks can be perfectly distributed among the processors such that no idle times occur. Similarly, $E_{low}(s)$ is the minimum energy consumption which results if no idle times consuming static power consumption occur. In this case, Equ. (6) simplifies to Equ. (10).*

Both lemmas show that the idle times are a crucial issue for the makespan and the energy consumption of a schedule.

4 Multicriteria decision problem

In this section, we broaden the optimization problem from Section 3 and consider a multicriteria decision problem which addresses time and energy together and whose solution combines scheduling algorithms and different ways of selecting the operational frequency, the result of which are called schedule execution modes.

4.1 Schedule execution modes

Algorithm 1 and Algorithm 2 are used to construct schedules A^T_f and A^E_f for all operational frequencies $f \in \mathcal{F}$ available. Usually, for different operational frequencies f, different schedules A^T_f as well as A^E_f result due to the non-linear dependence of the execution time of the tasks on the frequencies. The different schedules lead to different makespans and different energy consumptions.

There is the possibility to execute a schedule A^T_f or A^E_f generated for a specific operational frequency f with scaling factor s with a different operational frequency $\tilde{f}$ with scaling factor $\tilde{s}$. The usage of a different operational frequency $\tilde{f}$ does not change the task assignment of A^T_f or A^E_f to the processors, but may lead to a different makespan $T^{A_f}_{total}(\tilde{s})$ and a different energy consumption $E^{A_f}_{total}(\tilde{s})$. We denote the resulting task executions as **schedule execution modes**. For each schedule A^T_f or A^E_f generated by Algorithm 1 or Algorithm 2, there exist $n_F - 1$ additional schedule execution modes, one for each $\tilde{f} \in \mathcal{F}$, $\tilde{f} \neq f$. Additionally, frequency adaptation of the individual processors can be applied as described in Subsection 4.2, again leading to additional schedule execution modes.

4.2 Frequency adaptation

After a schedule has been computed by Algorithm 1 or 2 using frequencies f and $\tilde{f}$ as described above, the frequencies of the individual processors can be adjusted to reduce the resulting energy consumption. Let p_e be the makespan processor, which has the largest accumulated execution time, i.e., $T_{[1:n]}(p_e, s) = T^{\mathcal{A}}_{total}(s)$ for scaling factor s with corresponding operational frequency f. The processors $p \neq p_e$ exhibit idle times $T_{[1:n]}(p_e, s) - T_{[1:n]}(p, s)$ during which static power is consumed. The idle time of a processor $p \neq p_e$ may be reduced by using a smaller operational frequency $\tilde{f} < f$ for p and by executing the tasks assigned to p with this frequency. However, $\tilde{f}$ should not be too small, since the overall makespan should not be increased, i.e., $\tilde{f}$ should be selected such that $T_{[1:n]}(p_e, s) \geq T_{[1:n]}(p, \tilde{s})$ and that $\tilde{f}$ minimizes $T_{[1:n]}(p_e, s) - T_{[1:n]}(p, \tilde{s})$ among all frequencies available. Algorithm 3 shows the pseudocode of the frequency adaptation.

Since the usage of a smaller operational frequency usually leads to a smaller energy consumption, the frequency adaptation may lead to a reduction of the energy consumption without increasing the makespan. The potential of frequency adaptation increases with the operational frequency f selected by Algorithm 1. If f is the smallest operational frequency available, there is no potential for frequency adaptation. However, if f is large, more frequencies $\tilde{f}$ are available and for each processor p it can be tested whether a smaller energy consumption may result. The potential for savings of energy consumption is indicated by the modified lower bounds shown in the following lemma:

Lemma 3 *For the scheduling with frequency adaptation of the individual processors with available scaling factors $\mathcal{S} = \{s_1, \ldots, s_{n_F}\}$, the value*

$$E^{scal}_{low} = \sum_{x \in \mathcal{T}} \min_{s \in \mathcal{S}} E_x(s) \tag{11}$$

is a lower bound for the overall energy consumption.

Proof 3 *The minimum energy consumption of a single task $x \in \mathcal{T}$ under frequency scaling is $min_{s\in\mathcal{S}} E_x(s)$, Thus, $\sum_{s\in\mathcal{T}} min_{s\in\mathcal{S}} E_x(s)$ is the minimum total energy consumption under frequency scaling, if every task can be executed with an idividual frequency setting.*

The Lemma 3 hints at a potential energy improvement for schedule execution modes due to frequency scaling. Algorithm 3 systematically improves the energy values resulting in additional schedule execution modes. The different schedule execution modes can be represented in the criterion space introduced in the next Subsection 4.3 according to their makespan and energy consumption There is one entry for each $\tilde{f} \in \mathcal{F}$, $\tilde{f} \neq f$.

4.3 Decision problem and criterion space

The problem of finding an optimal schedule execution mode in the sense defined in Subsection 3.1 can be considered as a multicriteria decision problem considering both execution time and energy consumption together. For this problem, a decision space and a criterion space are defined as follows:

- The **decision space** represents all possible executions of the set of tasks $\mathcal{T}$ on p_{max} processors with individual frequency scaling.
- The **feasible set** $\mathcal{A}$ is a set of solutions which is a subset of the decision space that contains the task executions represented by a schedule execution mode A^T_f or A^E_f. The provision of schedule execution modes has been described in Subsections 4.4 and 4.5.
- The **criterion (or objective) space** is the image of the decision space under the objective function mapping, which are the makespan T and the energy consumption E. The criterion space can be represented by a diagram in which the x-axis denotes the makespan and the y-axis denotes the energy consumption. Each feasible schedule execution mode A^T_f or A^E_f is represented by a criterion value according to its makespan and its energy consumption.
- The image of $\mathcal{A}$ under the mappings $T : \mathcal{A} \to \mathbb{R}$ and $E : \mathcal{A} \to \mathbb{R}$ form the feasible set in the criterion space.

The set of feasible solutions $\mathcal{A}$ is built up according to Section 4.1 and 4.2 and its image in the criterion space is determined, such that a set of efficient solutions with respect to the makespan and the energy consumption can

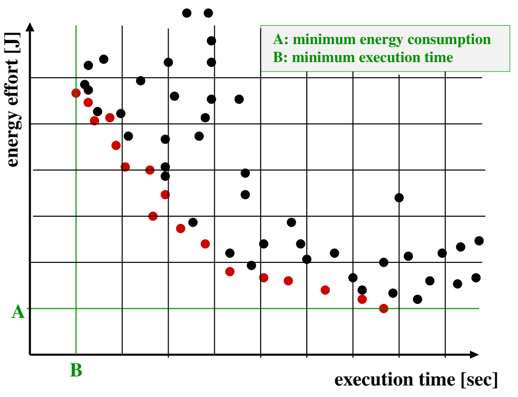


Figure 4: Illustration of the two-dimensional criterion space with the criterion values execution time and energy effort of schedule execution modes. There is a separate point for each schedule execution mode. The red points represent Pareto optimal points as defined in Definition 1.

be extracted. The definition of efficient solutions in the decision space and the related definition of nondominated points in the criteria space is given in Subsection 4.4.

An illustration of the two-dimensional criterion space (T^A, E^A) is given in Fig. 4, in which the horizontal axis corresponds to the execution time T^A and the vertical axis corresponds to the energy consumption E^A. Points in the criterion space (T^A, E^A) represent the criterion value for the specific schedule execution modes appraised with the criterion. The points to the left are better, since they correspond to a lower execution time. Lower points are better, since they correspond to the schedule execution modes with a lower energy consumption. Efficient alternatives, see Definition 1 in Section 4.4, are in the lower left corner and are depicted as red points. However, there is typically no best solution, since there may exist solutions with a lower execution time but a higher energy consumption and vice versa. The next Subsection 4.4 describes the multicriteria decision problem for time and energy and defines the notion of efficient (also called Pareto optimal) schedule execution modes.

4.4 Selection process

From the set of all schedule execution modes of all schedules A_f^T or A_f^E, we want to select those modes that lead to a best solution defined according to some optimization criterion, such as a minimum makespan or energy consumption or a combination of these. Not all schedule execution modes are candidates for the best solution. In fact, we can eliminate some of the schedule execution modes, since there are other modes that have a lower execution time and/or a smaller energy consumption in the criterion space. Next, we define **efficient** and **nondominated** schedule execution modes according to the general definition given in [3].

Definition 1 *A schedule execution mode A is called* **efficient (also called Pareto optimal)***, if there is no other schedule execution mode $\tilde{A}$ such that $T_{total}^{\tilde{A}} < T_{total}^{A}$ and $E_{total}^{\tilde{A}} < E_{total}^{A}$. If A is efficient, its entry $(T_{total}^{A}, E_{total}^{A}) \in \mathbb{R} \times \mathbb{R}$ in the criterion space is called* **nondominated point***. The set of efficient schedule execution modes is denoted by $\mathcal{A}_{\mathit{eff}}$. The set of all nondominated points is called the nondominated set. A schedule execution mode A_1* **dominates** *a schedule execution mode A_2, if $T_{total}^{A_1} \leq T_{total}^{A_2}$ and $E_{total}^{A_1} \leq E_{total}^{A_2}$.*

Thus, for the elements in $\mathcal{A}_{\text{eff}}$ there exists no alternative that has both a smaller execution time and a smaller energy consumption. The set of efficient solutions is sometimes also called Pareto set [3]. In Figure 4, the nondominated points are depicted in red, while the black points are points that are dominated by other points and therefore do not need to be considered further. All red points together represent the Parato set. All schedule execution modes that are not efficient can be excluded from the search for an optimal solution. For the computation of nondominated points and, thus, efficient schedule execution modes, we propose Algorithm 4.

The Algorithm 4 computes the set of efficient schedule execution modes, i.e., schedule execution modes with nondominated points in the criterion space, by first sorting all schedule execution modes according to increasing

Algorithm 4 Computation of the set of efficient schedule execution modes $\mathcal{A}_{\text{eff}}$.

```
1:  procedure SELECTSCHEDULE          ▷ Select nondominated schedule execution modes
2:      Let $\mathcal{A}$ with $\mid \mathcal{A} \mid = m$ be the set of feasible schedule execution modes generated;
3:      Sort $\mathcal{A}$ according to increasing execution time such that $T^{A_1}_{total} \leq \ldots \leq T^{A_m}_{total}$;
4:      $\mathcal{A}_{\text{eff}} = \{A_1\}$;
5:      $E_{min} = E^{A_1}_{total}$;
6:      for $(i = 2, \ldots, m)$ do
7:          if $(E^{A_i}_{total} \leq E_{min})$ then
8:              $\mathcal{A}_{\text{eff}} = \mathcal{A}_{\text{eff}} \cup \{A_i\}$;
9:              $E_{min} = E^{A_i}_{total}$;
10:         end if
11:     end for
12: end procedure
```

execution time and then traversing all schedule execution modes in the order of increasing execution time and deciding whether the next schedule execution mode is dominated by any of the schedule execution modes already visited. The runtime of this algorithm is $0(m \cdot \log m)$. The following lemma shows that Algorithm 4 is correct in the sense that it produces the efficient solutions.

Lemma 4 *If A_i is not included in $\mathcal{A}_{\text{eff}}$ by Algorithm 4, then A_i is dominated by some $A_j \in \mathcal{A}_{\text{eff}}$ with $j < i$.*

Proof 4 *If A_i is not included in $\mathcal{A}_{\text{eff}}$, it is $E^{A_i}_{total} > E_{min}$. E_{min} is caused by some $A_j, j < i$, i.e., $E_{min} = E^{A_j}_{total}, A_j \in \mathcal{A}_{\text{eff}}$. Since $j < i$, it is $T^{A_j}_{total} \leq T^{A_i}_{total}$ due to the sorting of the schedule execution modes. Thus it is $T^{A_j}_{total} \leq T^{A_i}_{total}$ and $E^{A_j}_{total} < E^{A_i}_{total}$, i.e., A_j dominates A_i.*

Thus, $\mathcal{A}_{\text{eff}}$ contains only solutions that are efficient. Moreover, all efficient schedule execution modes are included in $\mathcal{A}_{\text{eff}}$, since a schedule execution mode A_i is only excluded if another schedule execution mode A_j dominates A_i, i.e., if A_i is not efficient.

4.5 Schedule selection workflow

The procedure for determining efficient solutions for the multicriteria decision problem given in Section 4.4 is now formulated as **schedule selection workflow**. The workflow consists of several steps, including input specification, cost calculation, schedule construction and adaptation, and schedule selection according to a given objective function. The workflow is meant to provide a guideline for selecting an optimal schedule according to a given objective function. The steps to be executed are the following:

(1) **Input specification**: The specific scheduling problem is defined by a specific set $\mathcal{T}$ of tasks to be scheduled and the hardware specification including the maximum number p_{max} of processors and the set $\mathcal{F}$ of n_F frequencies supported by DVFS. The task set fulfills the property given in Section 3.1.

(2) **Cost determination**: For each task $x \in \mathcal{T}$, the cost information for the specific scheduling problem is determined. For $x \in \mathcal{T}$, the cost for frequency scaling factor s is the execution time $T_x(s)$ and/or the energy consumption $E_x(s)$. The way of determining the costs can be different, including the use of a theoretical computational model, physical measurements, model-based prediction, or source code analysis [11]. The cost information is stored in a table, giving the execution time $T_x(s)$ and the energy consumption $E_x(s)$ for all $x \in \mathcal{T}$ and $f \in \mathcal{F}$ with corresponding scaling factor s.

(3) **Schedule construction**: For each available frequency $f \in \mathcal{F}$ and each number p of processors, the scheduling algorithms Algorithm 1 and Algorithm 2 from Section 3 are used to construct schedules $A^T_{f,p}$ for time (T-schedule) resulting from Algorithm 1 and $A^E_{f,p}$ for energy (E-schedule) resulting from Algorithm 2. The total number of schedules constructed is $N_1 = p_{max} \cdot n_F \cdot 2$. For each of these schedules A, a makespan $T^A_{total}(s)$ and an overall energy consumption $E^A_{total}(s)$ is computed according to Equs. (3) and (6) where s is the corresponding scaling factor to the frequency f used.

(4) **Schedule execution modes**: Each schedule $A = A^T_{f,p}$ or $A = A^E_{f,p}$ constructed is executed with frequency f and with any other frequency $\tilde{f} \neq f$. This results in makespan $T^A_{total}(\tilde{s})$ and energy consumption $E^A_{total}(\tilde{s})$ where $\tilde{s}$ is the corresponding scaling factor to $\tilde{f}$. The resulting schedules executed with frequency $\tilde{f}$ are called schedule execution modes and are denoted as $A_{f,p}(\tilde{f})$ in the following. For each of the N_1 schedules constructed, n_F different costs result, which are stored in a table, i.e., $N_2 = N_1 \cdot n_F$ schedule execution modes are obtained.

(5) **Frequency adaptation**: For each schedule $A_{f,p}$ constructed in step (3) and each schedule execution mode $A_{f,p}(\tilde{f})$ from step (4), frequency adaptation is applied to the individual processors such that the overall makespan is not increased and the energy consumption of the individual processors is decreased as far as possible. The resulting adapted schedule execution mode for $A_{f,p}(\tilde{f})$ is denoted as $A^{adapt}_{f,p}(\tilde{f})$. By the adaptation, additional N_2 schedule execution modes are generated, resulting in $N_3 = N_2 \cdot n_F$ schedule execution modes, each represented by one dot in the criterion space diagram.

(6) **Determination of nondominated elements**: Steps (3), (4), (5) provide a set of schedule execution modes

$$\mathcal{A} = \{A_{f,p}(\tilde{f}) | \text{ for all } p, f, \tilde{f}\} \cup \{A^{adapt}_{f,p}(\tilde{f}) | \text{ for all } p, f, \tilde{f}\}.$$

The set $\mathcal{A}$ is the set of alternatives from which efficient solutions are to be chosen. The criteria for selecting an efficient solution are the cost information for time T^A_{total} and for energy E^A_{total} for each $A \in \mathcal{A}$. The subset $\mathcal{A}_{\text{eff}} \subseteq \mathcal{A}$ denotes the set of **efficient schedule execution modes**, which are defined to be all nondominated $A \in \mathcal{A}$ with respect to the criterion space $\{(T^A_{total}, E^A_{total}) | A \in \mathcal{A}\}$. Only the schedules in $\mathcal{A}_{\text{eff}}$ need to be considered for the following final schedule selection, since for each schedule $A \notin \mathcal{A}_{\text{eff}}$ there exists a schedule $A' \in \mathcal{A}_{\text{eff}}$ which provides a better solution than A.

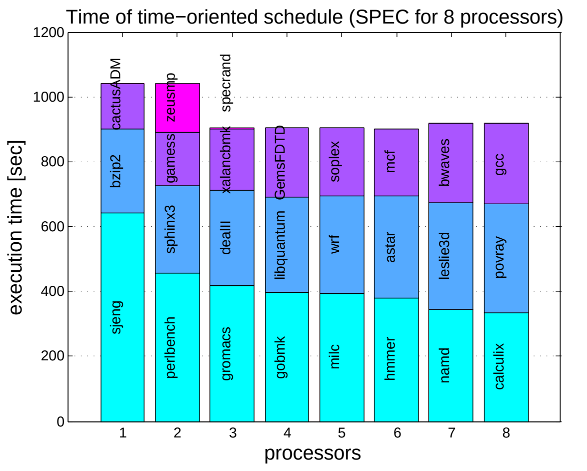
(7) **Schedule selection**: Given a specific optimization problem, such as minimum makespan or minimum energy consumption, the **optimal schedule execution modes** are selected from the set of efficient schedule execution modes $\mathcal{A}_{\text{eff}}$ by reducing the set to the specific optimization criteria. The result are several sets $\mathcal{A}^{opt}_1, \ldots, \mathcal{A}^{opt}_m \subset \mathcal{A}_{\text{eff}} \subset \mathcal{A}$ of optimal solutions each one for a specific optimization criterion. When considering the makespan in isolation, the schedule $A \in \mathcal{A}_{\text{eff}}$ with the minimum makespan is selected. When considering the energy consumption in isolation, the schedule $A \in \mathcal{A}_{\text{eff}}$ with the minimum energy consumption is selected. In these two cases, a single schedule remains to be considered. When considering a combination of makespan and energy consumption, several schedules $A \in \mathcal{A}_{\text{eff}}$ may need to be selected to avoid missing the optimal solution.

(8) **Schedule execution**: For a given scheduling problem defined in steps (1) and (2), the task program is executed according to the scheduling solutions pre-selected in step (7). If more than one schedule remains from step (7), the resulting makespans and energy consumptions are compared and the optimal solution is selected. Since the number of potential processors is included in every step (2)-(7), the optimal scheduling solution can be chosen for a specific number of processors.

Given the set of input tasks and the cost information from steps (1) and (2), the steps (3) – (6) provide the data basis for the selection of a solution of a specific optimization problem. Once the data is provided, step (7) can be applied several times with different optimization goals, which can be minimum execution time or energy consumption, but also a combination of the two. Several efficient solutions in the sense of Definition 1 may exist, which will be illustrated by the experimental evaluation in Section 5. The experimental evaluation includes an investigation of the dependence of solutions on the operational frequency and the number of processors used, see Section 5.3.

5 Experimental evaluation

In this section, we investigate the performance and energy behavior of T- and E-schedules resulting from Algorithm 1 and 2 and compare different schedule execution modes with respect to their energy-time behavior. For the experimental evaluation, we use runtime and energy measurements that have been obtained by executing the SPEC benchmark programs on an Intel Core i7-4770 with Haswell architecture.



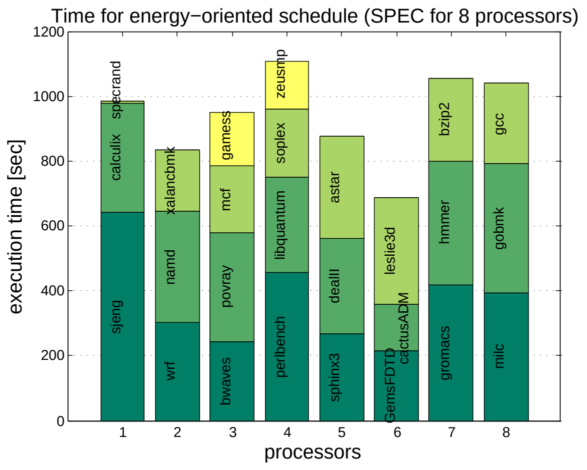


Figure 5: **Execution time**: Schedules from time-oriented scheduling (left) and energy-oriented scheduling (right) for SPEC benchmarks with Haswell timings for 3.4 GHz.

5.1 Experimental setting

For the experiments, we consider a set of basic tasks for which real measured data are taken as costs. As basic tasks we use the SPEC CPU2006 benchmarks consisting of integer and floating-point benchmarks, which are real programs covering a large variety of application areas. The suite has been developed with the goal to assess the performance of desktop systems and single-processor servers. The programs are sequential C, C++, or Fortran programs. The integer benchmarks include a compression program (bzip2), a C compiler (gcc), a video compression program, a chess game, or an XML parser. The floating point benchmarks include several simulation programs from physics, a speech recognition program, a ray-tracing program (povray), as well as programs from numerical analysis and a linear programming algorithm (soplex). More information about SPEC CPU2006 can be found in [7] and `www.spec.org`.

The performance data have been measured on an Intel Core i7-4770 with the Haswell architecture, which supports DVFS with 15 different operational frequencies $\mathcal{F} = \{0.8, 1.0, 1.2, 1.4, 1.5, 1.7, 1.9, 2.1, 2.3, 2.5, 2.7, 2.8, 3.0, 3.2, 3.4\}$, given in GHz. The benchmarks have been compiled with gcc 4.7.2. The programs from the SPEC benchmarks have been chosen, since they reflect, by design, the performance, energy, and execution time behavior of real-world programs. For the energy measurements, the likwid tool-set [30] has been used, see [22] for a detailed description of the measurement technique and a validation of the energy consumption data obtained. Based on these measurements, $P_{stat} = 5.5W$ has been determined using the model from [20] with linear regression.

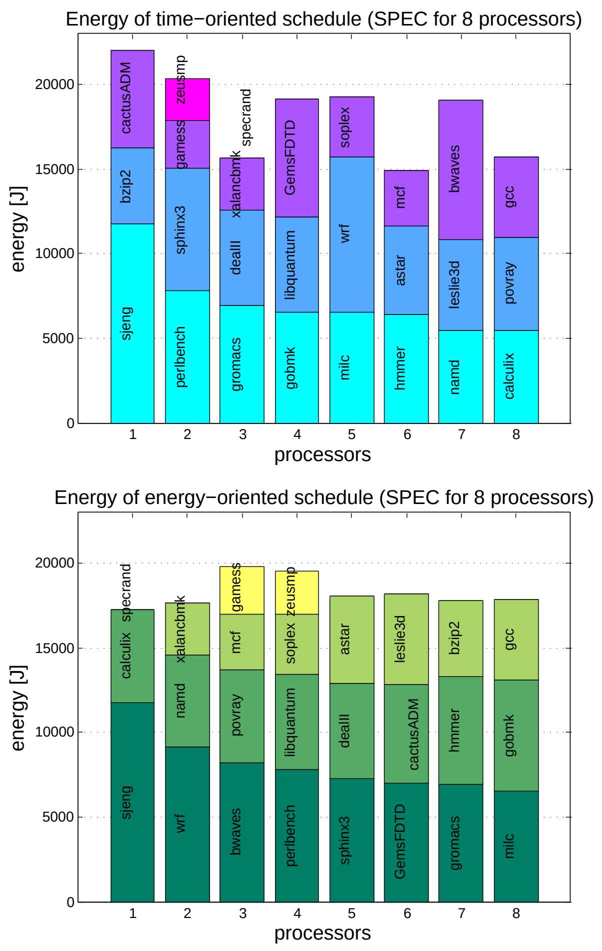


Figure 6: **Energy consumption**: Schedules from time-oriented scheduling (left) and energy-oriented scheduling (right) for SPEC benchmarks with Haswell timings using frequency 3.4 GHz.

5.2 Scheduling experiments

First, we show two example schedules and their time and energy behavior. Figure 5 shows two bar diagrams of schedules with execution times resulting from assigning the SPEC benchmark programs to eight processors operating at the largest frequency $f_{max} = 3.4$ GHz using a T-schedule from Algorithm 1 (left) and an E-schedule from Algorithm 2 (right). Figure 6 shows the corresponding energy consumption of the individual processors for the same schedules as in Fig. 5. From the figures it can be seen that the time-oriented scheduling leads to a quite even distribution of the execution times, whereas the energy-oriented scheduling is directed towards an even distribution of the energy values. It can be observed from Fig. 5 (right) that the energy-oriented scheduling yields a more uneven distribution of the execution times resulting in differently large idle times of the processors. Similarly, the time-oriented scheduling yields a more uneven distribution of the energy values Fig. 6 (left)).

Scheduling without adaptation Algorithm 1 and 2 provide 15 schedules each, one for each $f \in \mathcal{F}$ selected. The experiments have shown that the assignment functions A are actually different for different f. In our tests, we have then used all 15 schedules A each calculated for one f with different operational frequencies $\bar{f} \neq f$ according to Section 4.1. This results in 225 schedule execution modes with possibly different values for the makespan and the energy consumption for a fixed number of processors and a fixed task set. Figure 7 depicts the pairs of makespan and total energy resulting from the execution of the SPEC tasks on eight processors according to the 15 different schedules (T- schedule left, E-schedule right), each with 15 different operational frequencies $\bar{f}$. The horizontal axis gives the makespan of the schedule execution modes computed and the vertical axis shows the corresponding energy consumption. Each schedule execution mode contributes one dot in the criterion space. The corresponding operational frequencies are given as numbers in the diagrams next to the dots. It can be observed that the overall shape of the diagrams exhibits a u-shape, where each cluster of dots forms a line (from lower left upwards to the right) which belongs to one of the operational frequencies. In both diagrams, the frequencies between 1.5 and 2.1 GHz form a cluster with similar makespan and energy consumption. When comparing the left and the right diagram, the makespan as well as the energy consumption are higher when executing the tasks according to the energy-oriented schedules, which is indicated by dots with positions that are higher and more to the right. Thus, when looking for the schedule with the smallest makespan or the lowest energy consumption, these can be found in the left diagram. The following minimum values are obtained:

- The smallest makespan of 1008.7 seconds results for a T-schedule from Algorithm 1 with $f = 2.3$ GHz selected and using operational frequency $\bar{f}$ = 3.4 GHz. The corresponding energy consumption is 149253 Joule.
- The smallest energy consumption of 120155 Joule results for a T-schedule from Algorithm 1 with $f = 0.8$ GHz selected and using operational frequency $\bar{f}$ = 1.5 GHz. The corresponding makespan is 1542.6 seconds.

In contrast for the energy-oriented schedules (right diagram):

- The smallest makespan of 1028.7 seconds results for an E-schedule from Algorithm 2 with $f = 1.4$ GHz selected and using operational frequency $\bar{f}$ = 3.4 GHz.
- The smallest energy consumption of 122566 Joule results for an E-schedule from Algorithm 2 with $f = 1.4$ GHz selected and using operational frequency $\bar{f}$ = 1.5 GHz.

These numbers show that the schedule with the lowest makespan results for the highest operational frequency 3.4 GHz and the lowest energy consumption results for a medium frequency 1.4 GHz for both classes of schedules. Interestingly, the best solutions do not stem from the schedules obtained for an $f \in \mathcal{F}$ selected and using the same frequency $f = \bar{f}$ as operational frequency for the execution of the task program. Also it is interesting that the energy-oriented schedules lead to higher energy consumptions than the time-oriented schedules. This can be explained by the fact that idle times in the schedule contribute to the overall energy consumption proportionally to the value of P_{stat}, see Equ. (6), and that the energy-oriented schedules exhibit higher idle times, as it was illustrated by the examples in Figure 6.

Scheduling with adaptation Next we study the effect of frequency scaling described in Algorithm 3, see Subsection 4.2, which corresponds to step (5) in the schedule selection process, see Subsection 4.5. The diagrams in Fig. 7 depict the makespan-energy pairs for the schedules without adaptation, i.e., all processors use the same frequency. By applying the frequency adaptation for the individual processors to hide idle times as much as possible without increasing the overall makespan, a significant further reduction in the energy consumption can be obtained. This can be seen in Fig. 8, which zooms into the lower left rectangle of Fig. 7, where the efficient solutions can be found, and additionally depicts the makespan and the energy consumption of the schedules with individual frequency scaling (brown dots). The decrease in the energy consumption is especially large for the E-schedules. This is due to the fact that the energy-oriented scheduling leads to larger variations in the execution times of the individual processors, leaving more potential for the scaling. The following observations can be made for schedule execution modes with adaptation according to Algorithm 3.

- For the T-schedules generated by Algorithm 1 with frequency adaptation, the smallest energy consumption of 117555 Joule is obtained for a schedule with $f = 2.3$ GHz selected using operational frequency $\bar{f}$ = 2.3 GHz. The corresponding makespan is 1438.9 seconds.
- For the E-schedules generated by Algorithm 2 with frequency adaptation, the smallest energy consumption of 119038 Joule results for a schedule with $f = 1.0$ GHz using operational frequency $\bar{f}$ = 2.3 GHz. The corresponding makespan is 1517.1 seconds.

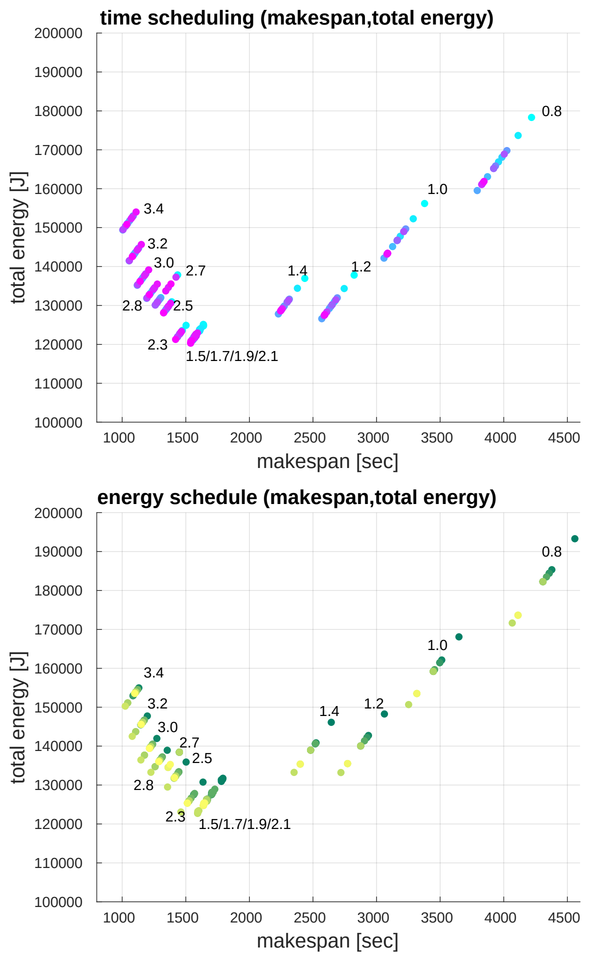


Figure 7: Pairs of (makespan, energy consumption) resulting from different schedules using different frequencies for time-oriented scheduling (left) and energy-oriented scheduling (right) for SPEC benchmarks on eight processors.

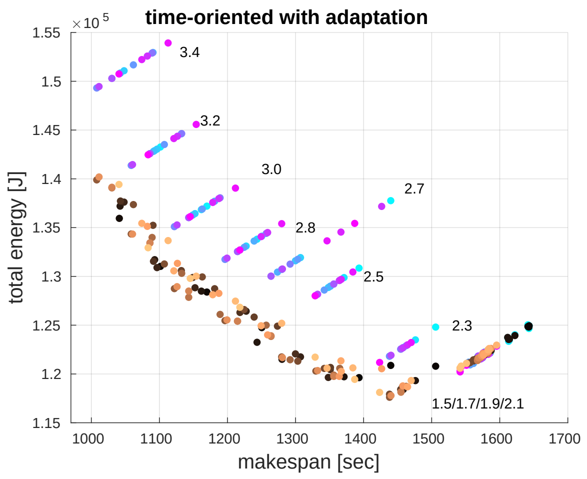
The results show that the time-oriented scheduling still leads to schedule execution modes with smaller energy consumption, although the difference between T-schedules and E-schedules has been reduced. When the optimization of both, the makespan and the energy consumption, is the goal of scheduling, then the dots depicted in the left lower rectangle of Fig. 8 left or right are the efficient solutions serving as good candidates.

The preceding experiments have used eight processors. Similar results have been obtained for other numbers of processors and other numbers of tasks. In general, it can be observed that the resulting idle times are smaller if the number of tasks is large compared to the number of processors, leaving less room for the frequency scaling of the individual processors, especially for the time-oriented scheduling.

5.3 Dependence on the frequency

In this subsection, we investigate and compare the makespan and energy consumption of schedules in dependence of the operational frequency. Figure 9 compares schedules that have been obtained by Algorithm 1 (T-schedule) and Algorithm 2 (E-schedule) for the SPEC benchmarks on 12 processors. The SPEC benchmark timings for 3.4 GHz have been used and the schedules resulting from Algorithm 1 and Algorithm 2 are tested for different operational frequencies. The left diagram shows the resulting makespan for the different frequencies. Fig. 10 shows the same information for 8 processors.

In Figures 9 and 10, it can be observed, that Algorithm 1 produces a schedule which has a smaller makespan than the schedule from Algorithm 2 for all frequencies. This could be expected, since Algorithm 1 uses the makespan



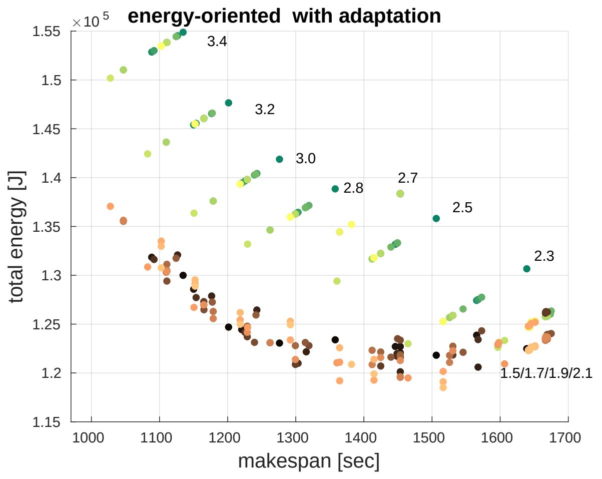


Figure 8: Energy improvement by frequency adaptation for time-oriented (left) and energy-oriented (right) scheduling for the most relevant operational frequencies.

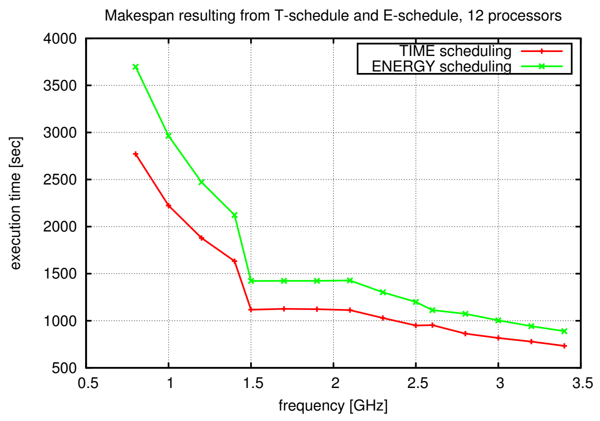
as objective function, whereas Algorithm 2 used the overall energy consumption as objective function. The same makespans result if frequency adaptation is used for the individual processors, since the frequency adaptation performed by the two algorithms does not effect the makespan processor. The right diagrams of Figures 9 and 10 show the resulting energy consumption of the schedules with and without frequency adaptation.

Without frequency adaptation, the schedule produced by Algorithm 1 always has a smaller energy consumption than the corresponding schedule produced by Algorithm 2 with the same frequency selected. This could be expected according to Lemma 1, which states that a smaller makespan leads to a smaller energy consumption if no frequency adaptation is used and since Algorithm 1 produces schedules with smaller makespans than Algorithm 2. However, taking frequency adaptation into account, the schedule produced by Algorithm 2 leads to a smaller energy consumption than the schedule produced by Algorithm 1 for larger operational frequencies. This can be explained by the fact that larger frequencies lead to more potential for frequency adaptation of the different processors and thus more energy saving.

For smaller operational frequencies the schedule produced by Algorithm 1 has a slightly smaller energy consumption than the schedule produced by Algorithm 2. For $f = 0.8$ GHz, the schedule with and without adaptation have the same energy consumption, since there is no possibility for reducing the frequency any further.

5.4 Dependence on the number of processors

We now consider the dependence of the makespan and the energy consumption of schedules generated by Algorithms 1 (T-schedule) and Algorithm 2 (E-schedule) on the number of processors employed, assuming that at most



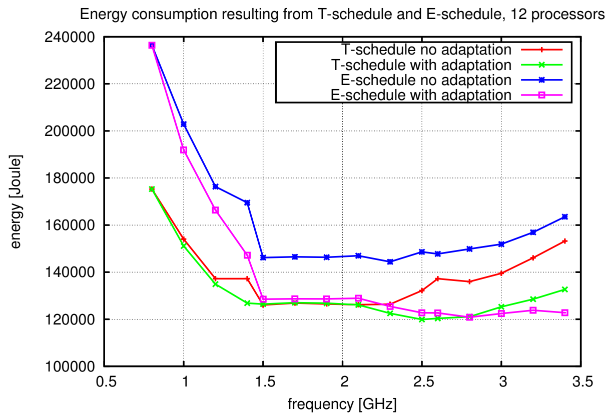
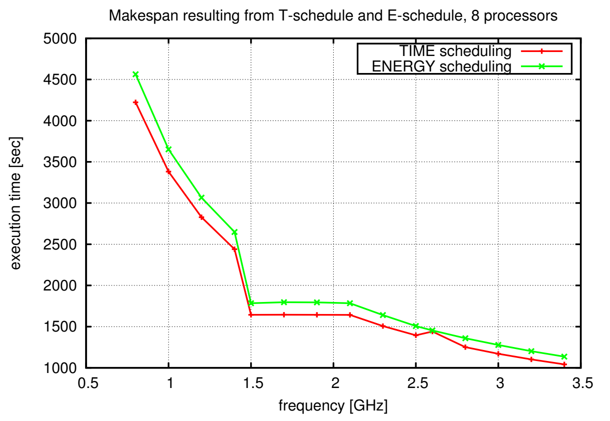


Figure 9: Comparison of time-oriented (T-schedule) and energy-oriented (E-schedule) scheduling with and without frequency adaptation for 12 processors: Left: resulting makespan for different frequencies. Right: resulting energy consumption for different frequencies.

p_{max} processors are available. Typically, the execution time decreases with the number of processors employed, but also there usually exists a saturation point of the number of processors for which the lowest possible execution time is provided, such that for a larger number than the saturation point a higher execution time results. For the energy consumption the growth behavior is different. For a growing number of processors, there is usually a higher overall power consumption. If the savings in execution time due to parallelism are large enough, then there is also a saving in energy. But if the savings of execution time is too small to compensate the higher power consumption, then the energy consumption might increase with the number of processors. Thus, there can also be an energy saturation point of the number of processors at which the decreasing behavior of the energy consumption is changed into an increasing behavior. However, this saturation points for the energy consumption can be different from the saturation points for the execution time.

Figure 11 illustrates the described behavior using the 26 SPEC benchmarks as tasks with measured execution times for 3.4 GHz for the makespan (left) and for the energy consumption (right), both for T-schedules and E-schedules without and with frequency adaptation. For the makespan, the results are similar as for the dependence on the frequency: T-schedules produced by Algorithm 1 lead to smaller makespans than E-schedules produced by Algorithm 2, until the saturation point at 14 processors is reached. In Figure 11 (left), it can be seen that for more than 14 processors, no further reduction in execution time will be achieved if more processors are employed and that T-schedules and E-schedules lead to the same makespan. In Figure 11 (right), it can be seen that for the energy consumption without frequency adaptation, T-schedules are better than the corresponding E-schedules for up to 16 processors. Beyond 16 processors, T-schedules and E-schedules cause the same energy consumption and the energy consumption increases if the number of processors is increased, that means the saturation point of the energy consumption is reached. With frequency adaptation, E-schedules have a smaller energy consumption than the corresponding T-schedules between 2 and 12 processors and T-schedules have a smaller energy consumption than



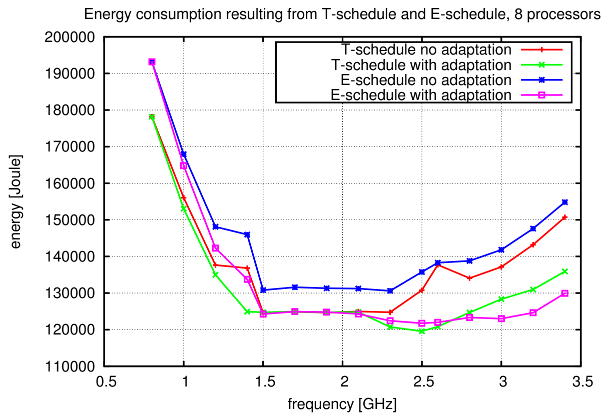


Figure 10: Comparison of time-oriented (T-schedule) and energy-oriented (E-schedule) scheduling with and without frequency adaptation for 8 processors: Left: resulting makespan for different frequencies. Right: resulting energy consumption for different frequencies.

E-schedules between 14 and 18 processors. For more than 18 processors, T-schedules and E-schedules produce very similar energy consumption.

Similar qualitative observations can be made for a larger number of tasks. Figure 12 shows the same information as Figure 11, now using 260 tasks with different numbers of processors. The tasks are obtained by generating 10 identical tasks for each of the 26 SPEC benchmarks. It can be observed that T-schedules lead to smaller makespans than the corresponding E-schedules for up to 160 processors. Beyond 160 processors, T-schedules and E-schedules have the same makespan, and the makespan is not reduced further if more processors are employed, showing that 160 processors is the saturation point for the makespan. Accordingly, without frequency adaptation, the energy consumption of T-schedules is smaller than the energy consumption of the corresponding E-schedules for up to 160 processors. Again, frequency adaptation has a larger effect for E-schedules than for T-schedules and for up to 128 processors, the resulting overall energy consumption for E-schedules is smaller than the energy consumption of the corresponding T-schedules. Then the situation changes and between 144 and 192 processors T-schedules with frequency adaptation have a slightly smaller energy consumption than the corresponding E-schedules. Beyond 192 processors, T-schedules and E-schedules lead to very similar energy consumption with adaptation, and the energy consumption increases with the number of processors, showing an energy saturation point at 192 processors. Comparing Figures 11 and 12, it can be observed that the different T-schedules and E-schedules with and without frequency adaptation show a very similar qualitative behavior. This can also be observed for other numbers of tasks. Especially, if the number of tasks per processor is very small, the makespan and energy consumption of T-schedules and E-schedules with and without frequency are very similar.

These observations show that the dependence of the schedule execution modes on the diverse parameters are quite complex. Thus, our approach to produce a large variety of schedule execution modes and compare their makespan and energy consumption in the criterion space for finding efficient solutions is a suitable way to handle

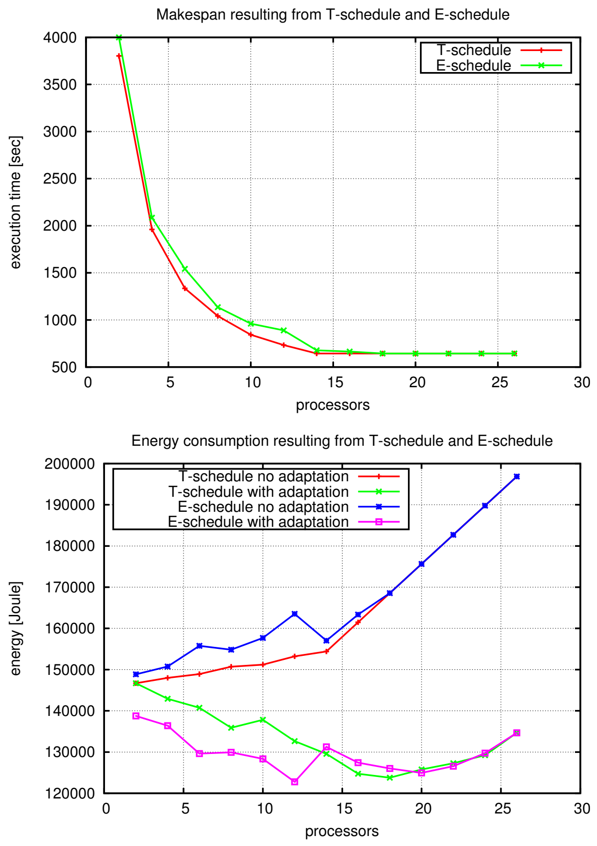


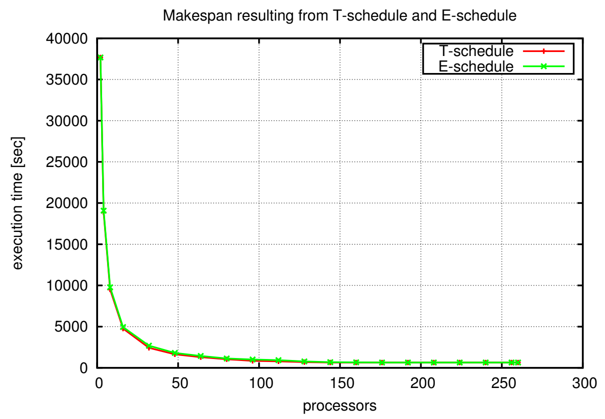
Figure 11: Execution time (left) and energy consumption (right) with different number of processors for 26 SPEC tasks.

this complexity.

6 Related Work

Energy saving is an important topic for HPC system, Cloud infrastructures, and data centers, see [10, 18] for a review of the state of the art in this area. Efficient architectures and scheduling frameworks have been proposed [32, 31]. Several online scheduling algorithms and resource provisioning techniques have been investigated especially for a usage in data centers [27, 19]. A survey of energy-cognizant techniques within the scheduler of the operating system (OS) is given in [33], with an emphasis on DVFS and Dynamic Power Management (DPM), thermal management and asymmetric multicore designs. However for DVFS, the techniques considered are meant for a dynamic adjustment of the frequency level during the execution of the applications by the OS. Energy measurement techniques, energy models and implications for scheduling algorithms have been considered in [22, 20], using randomly generated task sets for illustration. In this article, extended scheduling algorithms are used and the performance investigation is based on measured data for execution time and energy consumption.

Energy-efficient scheduling algorithms for sequential tasks have been proposed in [14, 15, 16], assuming different task dependencies and considering both continuous and discrete speed levels. In particular, the scheduling is investigated as combinatorial optimization problems, considering the problem of minimizing the makespan with energy consumption constraint and the problem of minimizing energy consumption with makespan constraint. Task scheduling and power allocation to tasks are done one after another, assuming that the power assignment can be done independently from the processor to which a task has been assigned; this is different from our approach, which assumes that the frequency of an individual processor is not changed during the execution of the task. Moreover, the focus of the works mentioned above lies on theoretical aspects of the scheduling algorithms, whereas our results concentrate on the practical implementation using time and energy measurements of real applications.



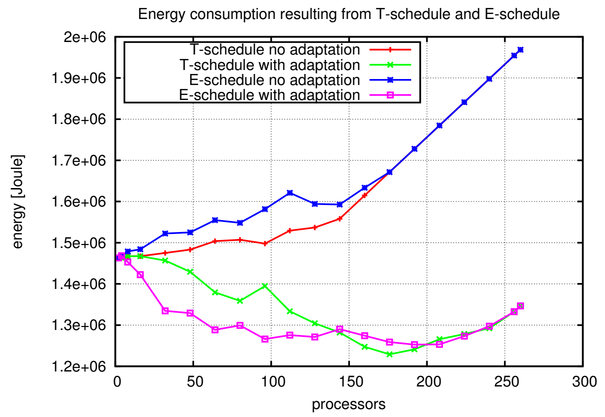


Figure 12: Execution time (left) and energy consumption (right) with the different number of processors for 260 SPEC tasks.

A detailed survey of theoretical aspects of the scheduling of task graphs using different energy models is given in [1]. The issue of selecting an optimal frequency for DVFS processors to reduce the energy consumption is addressed in [23]. In particular, it is assumed that the frequency for each task can be changed during its execution. Based on this assumption, the optimal frequency for each task is determined by a DVFS scheduling algorithm. In contrast, our approach makes the assumption that each task is executed with a fixed frequency, but different tasks may use different frequencies when executed on different processors. Two energy-aware scheduling algorithms for independent tasks have been proposed in [24]. The algorithms schedule tasks to the processors based on weighted aggregation cost function and then a task migration phase follows. Many other scheduling algorithms for DVFS environments have been proposed in the literature, see [5, 25, 29, 28]. An earlier version of this article has concentrated on an experimental evaluation of T-schedules and E-schedules [21].

When executing different tasks simultaneously on different cores of a multi-core system, inferences may occur due to simultaneous accesses to shared resources, such as shared caches or DRAM memory. The interference might be quite complex and are therefore difficult to capture by a static scheduling approach as the one proposed in this article. However, the interference may have an impact on the resulting makespan and energy consumption measured when executing a schedule on multi-core system and the resulting makespan and energy consumption measured at runtime could differ from the makespan and the energy consumption assumed by the static scheduling method. An in-depth analysis of the energy consumption of applications on multi-core systems exploiting the cache-aware roofline model [8] and using extensions for capturing the energy consumption and energy efficiency has been presented in [9]. Several power and energy consumption models for the core and the uncore domain of a multi-core system, also considering DVFS, are derived and are evaluated for matrix multiplication. Models for the performance, the power consumption and the energy efficiency on GPUs are provided in [17].

The task sets investigated in this article are independent tasks without ready times. Therefore idle times of processors can occur only after a processor has executed all tasks assigned to it. The situation would be more complex, if ready times for tasks were included. Then other scheduling algorithms would need to be employed to take the ready times into consideration, see e.g. [2]. If tasks had ready times, idle times could also occur between the execution of tasks assigned to a processor, in case that the next task to be executed by this processor, is not yet ready. These additional idle times might offer another possibility for frequency adaptation of the individual processors.

7 Conclusions

This article has considered time-oriented and energy-oriented scheduling algorithms which assign independent tasks to execution units in a list-scheduling like fashion exploiting a list of tasks sorted according to their execution time or their energy consumption, respectively. The experiments have shown that the same scheduling algorithms produces different schedules, when using the same number of execution units and the same set of tasks but different operational frequencies for measuring the task execution time and energy consumption. Also the T-schedules resulting from time-oriented scheduling and the E-schedules resulting from energy-oriented scheduling differ from each other even for the same frequency used for the measurements. Further investigations have concentrated on the effect when executing the schedules determined with different settings of the operational frequency. First, all schedules have been executed with all operational frequencies available. Then, the frequency of the individual processors are adapted to reduce idle times. The resulting execution time and energy consumption of the schedule executions, called schedule execution modes, have been used as criterion space for a multicriteria decision problem to optimize both execution time and energy consumption.

The variety of schedule execution modes provide a multitude of different values for the indicators makespan and energy consumption, even for a fixed set of tasks and a fixed set of processors. The experiments have shown that there exists a large variation for these indicators, which is best seen in the two-dimensional diagrams of the criterion space depicting the indicator makespan in the horizontal direction and the indicator energy consumption in the vertical direction. In the criterion space, the time-optimal solution is the leftmost dot, the energy-optimal solution is the lowest dot, and the solution having a good but not the optimal values for both indicators can be found in the leftmost lower rectangle. An interesting result is that the most efficient solutions concerning time and energy often come from the time-oriented scheduling rather than from the energy-oriented scheduling.

Acknowledgment

This work was performed within the Federal Cluster of Excellence EXC 1075 *MERGE Technologies for Multifunctional Lightweight Structures* supported by the German Research Foundation (DFG). This work is also supported by the German Ministry of Science and Education (BMBF), project *Self-Adaptation of Time-step-based Simulation Techniques on Heterogeneous HPC Systems* (SeASiTe), number 01IH16012A/B.

References

[1] G. Aupy, A. Benoit, P. Renaud-Goud, and Y. Robert. Energy-Aware Algorithms for Task Graph Scheduling, Replica Placement and Checkpoint Strategies. In S. U. Khan and A. Y. Zomaya, editors, *Handbook on Data Centers*, pages 37–80. Springer, 2015.

[2] J. Blazewicz. Deadline Scheduling of Tasks with Ready Times and Resource Constraints. *Inf. Process. Lett.*, 8(2):60–63, 1979.

[3] M. Ehrgott. *Multicriteria Optimization.* Springer-Verlag, 2005.

[4] H. Esmaeilzadeh, E. Blem, R. Amant, K. Sankaralingam, and D. Burger. Power challenges may end the multicore era. *Commun. ACM*, 56(2):93–102, February 2013.

[5] A. Fanfakh, J.-C. Charr, R. Couturier, and A. Giersch. Energy Consumption Reduction for Asynchronous Message-passing Applications. *J. Supercomput.*, 73(6):2369–2401, June 2017.

[6] A. Fedorova, J.C. Saez, D. Shelepov, and M. Prietol. Maximizing Power Efficiency with Asymmetric Multicore Systems. *Commun. ACM*, 52(12):48–57, December 2009.

[7] J.L. Hennessy and D.A. Patterson. *Computer Architecture - A Quantitative Approach (5. ed.)*. Morgan Kaufmann, 2012.

[8] A. Ilic, F. Pratas, and L. Sousa. Cache-aware Roofline Model: Upgrading the Loft. *IEEE Comput. Archit. Lett.*, 13(1):21–24, January 2014.

[9] A. Ilic, F. Pratas, and L. Sousa. Beyond the Roofline: Cache-Aware Power and Energy-Efficiency Modeling for Multi-Cores. *IEEE Transactions on Computers*, 66(1):1–1, 01 2017.

[10] S.U. Khan and A.Y. Zomaya, editors. *Handbook on Data Centers*. Springer-Verlag, 2015.

[11] M. Kühnemann, T. Rauber, and G. Rünger. A Source Code Analyzer for Performance Prediction. In *Proc. of 18th IPDPS, Workshop on Massively Parallel Processing (CDROM)*. IEEE, 2004.

[12] M. Kühnemann, T. Rauber, and G. Rünger. Performance Modelling for Task-Parallel Programs. In M. Gerndt, V. Getov, A. Hoisie, A. Malony, and B. Miller, editors, *Performance Analysis and Grid Computing*, pages 77–91. Kluwer, 2004.

[13] J. Leung, L. Kelly, and J.H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

[14] K. Li. Scheduling Precedence Constrained Tasks with Reduced Processor Energy on Multiprocessor Computers. *IEEE Transactions on Computers*, 61(12):1668–1681, Dec 2012.

[15] K. Li. Energy and Time Constrained Task Scheduling on Multiprocessor Computers with Discrete Speed Levels. *J. Parallel Distrib. Comput.*, 95(C):15–28, September 2016.

[16] K. Li. Energy-Efficient Task Scheduling on Multiple Heterogeneous Computers: Algorithms, Analysis, and Performance Evaluation. *IEEE Transactions on Sustainable Computing*, 1(1):7–19, Jan 2016.

[17] A. Lopes, F. Pratas, L. Sousa, and A. Ilic. Exploring GPU performance, power and energy-efficiency bounds with Cache-aware Roofline Modeling. 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2017.

[18] A.-C. Orgerie, M.D. de Assuncao, and Lurent Lefevre. A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. *ACM Computing Surveys*, 46(4):47:1–47:31, 2014.

[19] H. Qian, F. Li, R. Ravindran, and D. Medhi. Optimal Resource Provisioning and the Impact of Energy-Aware Load Aggregation for Dynamic Temporal Workloads in Data Centers. *IEEE Transactions on Network and Service Management*, 11(4):486–503, Dec 2014.

[20] T. Rauber and G. Rünger. Modeling and Analyzing the Energy Consumption of Fork-Join-based Task Parallel Programs. *Concurrency and Computation: Practice and Experience*, 27(1):211–236, 2015.

[21] T. Rauber and G. Rünger. Comparison of Time and Energy Oriented Scheduling for Task-based programs. In *Proc. of the 12th Int. Conf. on Parallel Processing and Applied Mathematics (PPAM)*, LNCS 10777. Springer, 2018.

[22] T. Rauber, G. Rünger, M. Schwind, H. Xu, and S. Melzner. Energy Measurement, Modeling, and Prediction for Processors with Frequency Scaling. *The Journal of Supercomputing*, 2014.

[23] N.B. Rizvandi, J. Taheri, and A.Y. Zomaya. Some Observations on Optimal Frequency Selection in DVFS-based Energy Consumption Minimization. *J. Parallel Distrib. Comput.*, 71(8):1154–1164, August 2011.

[24] M. Sajid, Z. Raza, and M. Shahid. Energy-efficient scheduling algorithms for batch-of-tasks (BoT) applications on heterogeneous computing systems. *Concurrency and Computation: Practice and Experience*, 28(9):2644–2669, 2016.

[25] P. Sanjeevi and P. Viswanathan. NUTS Scheduling Approach for Cloud Data Centers to Optimize Energy Consumption. *Computing*, 99(12):1179–1205, December 2017.

[26] E. Saxe. Power-efficient software. *Commun. ACM*, 53(2):44–48, 2010.

[27] M. Shojafar, C. Canali, R. Lancellotti, and S. Abolfazli. An Energy-aware Scheduling Algorithm in DVFS-enabled Networked Data Centers. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2*, CLOSER 2016, pages 387–397, Portugal, 2016. SCITEPRESS - Science and Technology Publications, Lda.

[28] X. Tang and W. Tan. Energy-Efficient Reliability-Aware Scheduling Algorithm on Heterogeneous Systems. *Sci. Program.*, 2016:14–, March 2016.

[29] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li. An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. *J. Grid Comput.*, 14(1):55–74, March 2016.

[30] J. Treibig, G. Hager, and G. Wellein. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In *Proc. of 39th International Conference on Parallel Processing Workshops*, ICPP '10, pages 207–216. IEEE Computer Society, 2010.

[31] V. Villebonnet, G. Da Costa, L. Lefevre, J. M. Pierson, and P. Stolf. Energy Aware Dynamic Provisioning for Heterogeneous Data Centers. In *28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 206–213, Oct 2016.

[32] V. Villebonnet, G. Da Costa, L. Lefvre, J.-M. Pierson, and P. Stolf. Big, Medium, Little: Reaching Energy Proportionality with Heterogeneous Computing Scheduler. *Parallel Processing Letters*, 25, 09 2015.

[33] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. Survey of Energy-Cognizant Scheduling Techniques. *IEEE Transactions on Parallel and Distributed Systems*, 24(7):1447–1464, July 2013.