Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

ANALYTICAL MODELING AND SIMULATION OF THE ENERGY CONSUMPTION OF INDEPENDENT TASKS

Thomas Rauber

University Bayreuth
Computer Science Department
D-95440 Bayreuth, GERMANY

Gudula Rünger

Chemnitz University of Technology
Computer Science Department
D-09107 Chemnitz, GERMANY

ABSTRACT

The estimation and evaluation of the energy consumption of computers is becoming an important issue. In this article, we address the question how the energy consumption for computations can be captured by an analytical energy consumption model. In particular, we address the possibility to reduce the energy consumption by dynamic frequency scaling and model this energy reduction in the context of task execution models. We demonstrate the use of the model by simulating task executions and their energy consumption.

1 INTRODUCTION

Computers consume a significant amount of energy while they are running and producing results. Until recently, energy consumption has not been a major concern, and software has usually been built such that the computations are executed as fast as possible, leading to a minimal execution time of the program. This approach has especially been used in parallel computing where a minimal program execution time is an important issue. Thus, many programming techniques and execution modes have been proposed with the goal to structure the program computations and memory accesses such that a fast execution time results. Accordingly, for parallel or distributed programs the metrics used to evaluate program performance, such as speedup and efficiency, has been solely based on program execution times, no matter how much energy is consumed.

In the era of climate change, environmental concerns make it necessary to extend the considerations of program efficiency and to include energy consumption into the efficiency calculations (Saxe 2010). This is especially important for large complex software systems to be used in many areas of research and industry. In particular, it is important to develop suitable programming techniques for the design of programs which can exploit a hardware system such that their computations are performed with low energy consumption while guaranteeing good performance.

An important low-power design technique that is provided for many commodity processors is the dynamic voltage-frequency scaling (DVFS) technique, which enables processors to dynamically adjust voltage and frequencies of the entire processor or of individual cores aiming at a reduction of power consumption. Reducing the voltage leads to a smaller energy consumption. However, longer computation times may result due to the reduced frequency. In order to exploit frequency scaling for calculated energy saving, the influence of that technique has to be studied at the application programmer's level.

An effective way to structure parallel programs is the use of tasks (N'takpé, Suter, and Casanova 2007; Vydyanathan, Krishnamoorthy, Sabin, Catalyurek, Kurc, Sadayappan, and Saltz 2009), which can be generated statically or dynamically. A task-based programming approach enables a structuring of a parallel program according to the needs of an application algorithm and allows the use of load balancing and scheduling techniques (Hoffmann and Rauber 2011) for an efficient exploitation of the sequential or parallel hardware platform. It has been shown that a task-based programming model can also be advantageous for restructuring programs in the context of reducing energy consumption (Lee and Zomaya 2009; Zhuo and Chakrabarti 2008).

In this article, we exploit an analytical energy model simulating the energy consumption effect of frequency scaling. The model is applied to the execution of sequential tasks that may be executed concurrently to each other, resulting in a parallel execution. The resulting energy consumption for different processor assignments is investigated. For concurrently executed tasks, frequency scaling factors are derived that lead to a minimum energy consumption for a specific parallel program. In previous work (Rauber and Rünger 2011), we have applied the model to parallel tasks where each task can be executed by a set of processors. In Rauber and Rünger (2012), it has been shown that the energy model is well suited for modeling the energy consumption of current multicore processors with DVFS technique. This article contributes an experimental investigation of the modeling technique by simulations for a large number of randomly generated task sets with different distributions of the task execution times. Different scaling factors are used and the resulting energy consumption and execution times are compared.

The article is organized as follows: Section 2 resumes the task-parallel programming model and Section 3 presents the energy model used for simulating the energy consumption of task executions based on frequency scaling factors. Section 4 derives energy consumption functions for the parallel execution of tasks and presents corresponding frequency scaling factors. Section 5 presents an experimental evaluation of the energy model using several different distributions of the task execution times. Section 6 discusses related work and Section 7 concludes the article.

2 TASK-BASED PROGRAMMING MODEL

A task-based programming model assumes that the parallel program to be executed consists of a set of tasks that can be executed by any of the processors or cores provided by the given execution platform. For a parallel execution, several tasks can be executed in parallel, each on a separate processor or core, if they are independent. A processor can fetch the next task for execution as soon as it becomes idle. Task-based executions are integrated in many modern execution environments, such as OpenMP 3.0 and parallel languages, such as Chapel or X10.

The fork-join pattern is a possibility to express sections of independent tasks to be executed in parallel. At each point of program execution, new independent tasks can be created (fork) and the creating task may wait until all these tasks will be terminated (join), see Fig. 1 (left) for an illustration of the fork-join pattern. This includes an implicit barrier operation at the join operation involving all tasks of a fork-join construct.

In this article, the energy consumption of task-based programs with parallel executions of independent tasks is modeled. A homogeneous parallel platform with p identical processing units (processors or cores) is considered. Assuming that the program consists of a set $\mathscr{T}$ of tasks, the overall execution time depends on the execution time of the individual tasks $T \in \mathscr{T}$ and the coordination and waiting times of the tasks. The execution time of a task $T \in \mathscr{T}$ is given as a cost function C_T. This captures general task models as they are used by many modern libraries and languages, such as OpenMP, X10, Fortress, and Chapel. The cost function C_T can express the actual execution time on a specific hardware platform, which can be a measured time in seconds or a predicted time. More abstract functions depending on the needs of the application programmer can be used. Usually, the execution time also depends on other parameters, such as a problem size or parameters of the parallel execution platform. In the following, we consider a specific problem instance and, thus, these other parameters can assumed to be fixed.

The execution time for the entire parallel program consisting of a task set $\mathscr{T}$ is built up from the functions $C_T, T \in \mathscr{T}$, according to the structure of the tasks and the processor assignment used. The execution time for a set of tasks coordinated by a fork-join pattern $T_{\text{fork-join}}$ is described by the formula

$$C_{T_{\text{fork-join}}} = \max_{i=1,\dots,p} C_{T_i},$$

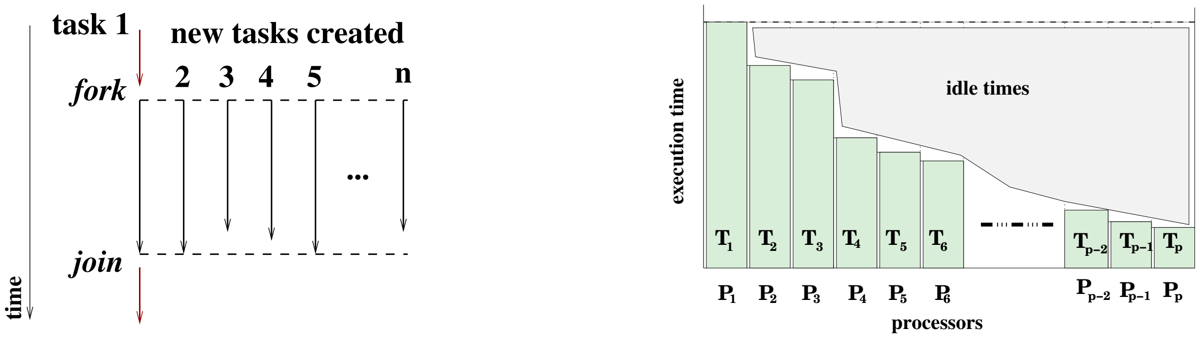


Figure 1: Left: illustration for the execution of a fork-join based task-parallel program. Right: illustration of idle times resulting from different task execution times.

assuming that p tasks $T_1, \ldots, T_p$ are created (fork) by a parent task T_0 and that a join requires a barrier synchronization, waiting for all tasks $T_1, \ldots, T_p$ to be completed before continuing. Thus, the execution time is dominated by the execution time of the task T_m with the longest execution time, i.e., $C_{T_m} = \max_{i=1,\ldots,p} C_{T_i}$. Waiting time may occur, since the processors executing other tasks must wait for the completion of T_m.

3 FREQUENCY-SCALING ENERGY MODEL

To capture the energy consumed by a processor, we use a well-accepted energy model that has already been applied to embedded systems (Zhuo and Chakrabarti 2008), to heterogeneous computing systems (Lee and Zomaya 2009), or to shared-memory architectures (Korthikanti and Agha 2010). This section summarizes the energy model according to Zhuo and Chakrabarti (2008) and enriches the model by considering leakage power and task execution. In the energy model used, the power consumption of a processor consists of the *dynamic* power consumption P_{dyn}, which is related to the switching activity and the supply voltage, and the static power consumption P_{static}, which captures the leakage power consumption as well as the power consumption of peripheral devices like the I/O subsystem (Jejurikar, Pereira, and Gupta 2004). The dynamic power consumption is approximated by

$$P_{dyn} = \alpha \cdot C_L \cdot V^2 \cdot f \tag{1}$$

where α is the switching probability, C_L is the load capacitance, V is the supply voltage, and f is the operational frequency. For many years, the dynamic power consumption has represented the predominant factor in CMOS power consumption, but for recent technologies, leakage power has an increasing impact and represents roughly 20 % of power dissipation in current processor designs (Kaxiras and Martonosi 2008). Modeling leakage power is difficult, since it consists of several components, including sub-threshold leakage, reverse-biased-junction leakage, gate-induced-drain leakage, gate-oxide leakage, gate-current leakage, and punch-through leakage. A simplified model for leakage power presented in Butts and Sohi (2000) models the static power consumption due to leakage power as

$$P_{static} = V \cdot N \cdot k_{\text{design}} \cdot I_{\text{leak}}$$

where V is the supply voltage, N is the number of transistors, k_{design} is a design dependent parameter, and I_{leak} is a technology-dependent parameter. In the following, similar to (Zhuo and Chakrabarti 2008), we make the simplified assumption that P_{static} is independent of the scaling factor s. This is justified by the close match between the data sheet curves of real DVFS (Dynamic Voltage-Frequency Scaling) processors and the analytical curves obtained by using this assumption, see (Zhuo and Chakrabarti 2008). Moreover, it should be noted that P_{dyn} from Equ. 1 does not capture the energy consumption of memory accesses or I/O. Therefore, the model considered is especially suited for non-memory intensive programs. These abstractions are made to keep the energy model as simple as possible to demonstrate the essentials of the analytical derivation technique presented in the following more clearly. The general methodology could also be applied to a more complex energy model that may additionally capture other sources of energy consumption.

3.1 Frequency Scaling Factors

The operational frequency f depends linearly on the supply voltage V, i.e., $V = \beta \cdot f$ with some constant β. This equation can be used to study the change of the dynamic voltage with respect to various frequency values. Reducing the frequency by a scaling factor s, i.e., using a different frequency value $\tilde{f} = s^{-1} \cdot f$ with $s \geq 1$, leads to a decrease of the dynamic power consumption. This can be seen by using Equ. (1) with $\tilde{f}$, resulting in the following equation for the dynamic power consumption:

$$\tilde{P}_{dyn} \quad = \alpha \cdot C_L \cdot V^2 \cdot \tilde{f} = \alpha \cdot C_L \cdot \beta^2 \cdot \tilde{f}^3 = \alpha \cdot C_L \cdot V^2 \cdot f \cdot s^{-3} \quad = s^{-3} \cdot P_{dyn} \quad . \tag{2}$$

According to Formula (2), the dynamic power is decreased by a factor of s^{-3} when reducing the frequency by a factor of s. In the following, s is used as a parameter and the dynamic power consumption with scaling factor s is denoted by $P_{dyn}(s)$. The rest of this section summarizes how to model the energy consumption of a single task executed sequentially on one processor.

3.2 Energy Consumption for a Sequential Execution of Tasks

Frequency scaling influences the execution time and the energy consumption of task executions. The sequential execution time C_T of a task $T \in \mathscr{T}$ increases linearly with s, resulting in the execution time $C_T \cdot s$, when the frequency is reduced by a factor of s. Using the scaling factor s as parameter of the energy consumption (which is the power consumption, measured in Watt, multiplied by the execution time), the dynamic energy consumption of the task T for one processor can be modeled as:

$$E^T_{dyn}(s) \quad = P_{dyn}(s) \cdot (C_T \cdot s) = s^{-3} \cdot P_{dyn}(1) \cdot (C_T \cdot s) = s^{-2} \cdot E^T_{dyn}(1) \tag{3}$$

with $E^T_{dyn}(1) = P_{dyn}(1) \cdot C_T$. Analogously, the static energy consumption can be modeled as:

$$E^T_{static}(s) \quad = P_{static} \cdot (C_T \cdot s) = s \cdot E_{static}(1) \tag{4}$$

with $E_{static}(1) = P_{static} \cdot C_T$. According to Equs. (3) and (4), the total energy consumption for the sequential execution of T on one processor is:

$$E^T(s) \quad = E^T_{dyn}(s) + E^T_{static}(s) = (s^{-2} \cdot P_{dyn}(1) + s \cdot P_{static}) \cdot C_T \tag{5}$$

expressing explicitly the dependence of the energy on the scaling factor s.

3.3 Optimal Frequency Scaling Factor

The optimal scaling factor minimizing the sequential execution of a task can be obtained by considering

$$Q(s) = s^{-2} \cdot P_{dyn}(1) + s \cdot P_{static} \tag{6}$$

of $E^T(s)$ in Equ. (5). Since C_T in Equ. (5) is independent of the scaling factor s (Zhuo and Chakrabarti 2008), the value that minimizes $Q(s)$ also minimizes $E^T(s)$. The function $Q(s)$ in Formula (6) is convex, because its second derivative $Q''(s)$ exists and $Q''(s) \geq 0$. Thus, the optimal scaling factor can be obtained by setting $Q'(s) = -2 \cdot P_{dyn}(1)/s^{-3} + P_{static}$ to zero. Hence, the optimal scaling factor minimizing the energy consumption $E^T(s)$ is

$$s_{opt} = \left(\frac{2 \cdot P_{dyn}(1)}{P_{static}} \right)^{1/3} . \tag{7}$$

Assuming that $P_{dyn}(1)$ is independent of the computations performed, s_{opt} depends only on the characteristics of the given processor. For a processor with $P_{static} = 4W$ and $P_{dyn}(1) = 20W$, $s_{opt} = 2.15$ results.

It should be noted that, in practice, only a finite number of frequency scaling factors is available for typical DVFS-enabled processors. Therefore, s_{opt} must be rounded to the nearest scaling factor available. This common methodology to treat scaling factors as arbitrary real numbers for the calculation and then round the result to a realistic discrete number applies to all scaling factors computed throughout the article.

The scaling factor s_{opt} from Equ. (7) minimizes the energy consumption of a single sequential task. This scaling factor can also be used for multiple tasks. However, the computation of s_{opt} does not yet take waiting times for concurrently executed tasks into consideration. Such waiting times may occur at join points when the tasks to be synchronized by a barrier have different execution times and cause idle times of the processors. The next section shows that in this context, other scaling factors can lead to even smaller energy values.

4 FREQUENCY SCALING FOR CONCURRENT TASKS

In this section, we investigate the energy consumption in the case that different scaling factors are used for the concurrent execution of different tasks. Section 4.1 states that in order to obtain a minimum energy consumption, the scaling factors for the different processors should be selected such that no idle times occur. Section 4.2 computes the resulting energy consumption for two concurrently executed tasks, and Section 4.3 generalizes the result to an arbitrary number of concurrent tasks.

4.1 Energy Optimal Scaling Factors

For the concurrent execution of tasks $T_1,\ldots,T_n$, $n \geq 2$, executed in parallel on n different processors $P_1,\ldots,P_n$, each processor executing one task may use a different frequency scaling factors $s_1,\ldots,s_n$ for the execution of its corresponding task. According to the fork-join semantics, all processors are busy for the same amount of time before they can start another computation. The time for processor P_i is the execution time for the task T_i and the idle time I_i elapsed before all other processors finish their execution, see Fig. 1 (right) for an illustration. Thus, the time for P_i is

$$C_{P_i}(s_i) = C_{T_i} \cdot s_i + I_i \,. \tag{8}$$

The energy optimal execution is achieved by a set of frequency scaling factors ${s_1}^*,\ldots,{s_n}^*$ for which the overall energy consumption

$$E^{T_1\|\ldots\|T_n}({s_1}^*,\ldots,{s_n}^*) = \sum_{i=1}^{n}(C_{T_i} \cdot Q(s_i) + I_i \cdot P_{static}) \tag{9}$$

is minimal. In Rauber and Rünger (2012), it has been shown that such a solution has the property $I_i = 0$ for $i = 1,\ldots,n$. This results when using scaling factors ${s_1}^*,\ldots,{s_n}^*$ for tasks $T_1,\ldots,T_n$ with ${s_i}^* \geq 1$ for $i = 1,\ldots,n$ such that $C_{T_i} \cdot {s_i}^* = C_{T_j} \cdot {s_j}^*$ for all pairs of tasks $T_i, T_j, i, j \in \{1,\ldots,n\}$, with sequential execution time $C_{T_i}, i = 1,\ldots,n$.

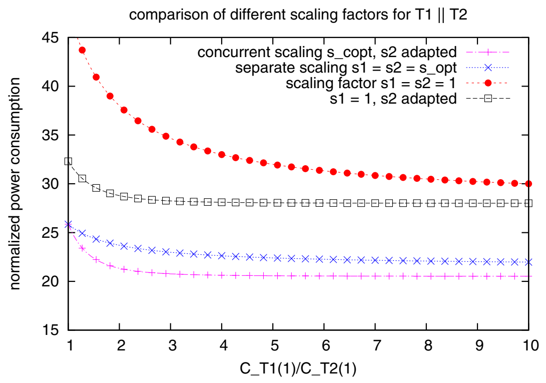
4.2 Two Concurrently Executed Tasks

This section exploits the results of Section 4.1 for the execution of optimal scaling factors for two concurrent tasks T_1 and T_2. These scaling factors s_1 and s_2 have to fulfill $s_1 \cdot C_{T_1} - s_2 \cdot C_{T_2} = 0$ and, thus:

$$s_2 = s_1 \cdot \frac{C_{T_1}}{C_{T_2}}. \tag{10}$$

Using Equ. (10) in the energy consumption equation for $T_1||T_2$

$$E^{T_1\|T_2}(s_1,s_2) = (s_1^{-3} \cdot P_{dyn}(1) + P_{static}) \cdot s_1 \cdot C_{T_1} + (s_2^{-3} \cdot P_{dyn}(1) + P_{static}) \cdot s_2 \cdot C_{T_2} \tag{11}$$



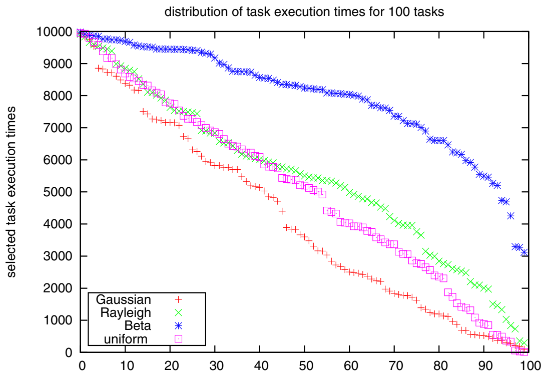


Figure 2: Left: Comparison of the normalized energy consumption $E^{T_1\|T_2}(s_1,s_2)/C_{T_1}$ of two concurrent tasks using different scaling factors for an example processor with $P_{static} = 4W$ and $P_{dyn}(1) = 20W$. Right: Distribution of the task distribution times for a task set with 100 tasks, selecting the task execution times between 1s and 10000s using different distribution functions.

leads to

$$E^{T_1\|T_2}(s_1,s_2) = s_1^{-2} \cdot P_{dyn}(1) \left(C_{T_1} + \frac{C_{T_2}^3}{C_{T_1}^2} \right) + 2s_1 \cdot P_{static} \cdot C_{T_1}. \tag{12}$$

The energy function (12) is differentiable and has a minimum at

$$s_{copt} = \sqrt[3]{\frac{P_{dyn}(1)}{P_{static}} \left(1 + \frac{C_{T_2}^3}{C_{T_1}^3} \right)} \tag{13}$$

which is computed by setting the derivative

$$\frac{d}{d_{s_1}} E^{T_1\|T_2}\left(s_1, s_1 \cdot \frac{C_{T_1}}{C_{T_2}}\right) = -2s_1^{-3} P_{dyn}(1) \left(C_{T_1} + \frac{C_{T_2}^3}{C_{T_1}^2} \right) + 2P_{static} \cdot C_{T_1}$$

to zero. Equation (13) is a generalization of s_{opt} in Equ. (7), and for $C_{T_1} = C_{T_2}$, Equ. (13) simplifies to Equ. (7).

Figure 2 (left) shows the normalized energy consumption for $T_1||T_2$ according to Equ. (11) for different scaling factors s_1 and s_2 and varying values of C_{T_1}/C_{T_2}. No specific values are assumed for C_{T_1} and C_{T_2}, and the x-axis depicts different relative sizes of C_{T_1} compared to C_{T_2}. The energy consumption shown is normalized with respect to C_{T_1}, i.e., the values $E^{T_1\|T_2}(s_1,s_2)/C_{T_1}$ are depicted. Four different scaling factors are compared; (i) concurrent scaling with s_{copt} according to Equ. (13) and s_2 chosen according to Equ. (10); (ii) separate scaling with $s_1 = s_2 = s_{opt}$ according to Equ. (7); (iii) no scaling, i.e., $s_1 = s_2 = 1$, and (iv) no scaling for s_1 and s_2 adapted according to Equ. (10). The figure illustrates that the smallest amount of energy is consumed for the scaling factors according to Equ. (13) and (10). The resulting energy consumption is smaller than the energy consumption resulting when using the optimal scaling factor s_{opt} for a task in isolation. The reason is that s_{opt} does not take the waiting time into consideration. For $C_{T_1} = C_{T_2}$, cases (i) and (ii) result in the same energy consumption, since $s_{copt} = s_{opt}$ and $s_2 = s_{opt}$ in that case. The other two cases lead to a much larger energy consumptions.

4.3 Arbitrary Number of Tasks

The result obtained for choosing the optimal scaling factors for two concurrently executed sequential tasks is now generalized to an arbitrary number of tasks. Let $T_1,\dots,T_n$ be a set of independent tasks that have been generated by a fork statement and that are executed concurrently on n processors. We assume that the tasks are ordered in decreasing order of their sequential execution time C_{T_i}, $i=1,\dots,n$, i.e., T_1 is the task with the largest execution time. According to Equ. (10), the scaling factors for each task $T_i \in \{T_2,\dots,T_n\}$ are set to

$$s_i = s_1 \cdot \frac{C_{T_1}}{C_{T_i}} \tag{14}$$

to get an optimal energy result. Using Equ. (14) for s_i results in the following total energy consumption:

$$E^{T_1\|\dots\|T_n}(s_1,\dots,s_n) == s_1^{-2} \cdot P_{dyn}(1)\left(C_{T_1} + \sum_{i=2}^{n} \frac{C_{T_i}^3}{C_{T_1}^2}\right) \quad + n \cdot s_1 \cdot P_{static} \cdot C_{T_1}\,. \tag{15}$$

To compute the minimum, the derivative

$$\frac{d}{d_{s_1}} E^{T_1\|\dots\|T_n}(s_1,\dots,s_n) = -2s_1^{-3} P_{dyn}(1)\left(C_{T_1} + \sum_{i=2}^{n} \frac{C_{T_i}^3}{C_{T_1}^2}\right) + n \cdot P_{static} \cdot C_{T_1}$$

is considered and set to zero. This yields that $E^{T_1\|\dots\|T_n}$ is minimized, if the scaling factor

$$s_{copt}(n) = \sqrt[3]{\frac{2}{n}\frac{P_{dyn}(1)}{P_{static}}\left(1 + \sum_{i=2}^{n} \frac{C_{T_i}^3}{C_{T_1}^3}\right)} \tag{16}$$

is used for s_1. The scaling factors s_i, $i=2,\dots,n$, are then determined according to Equ. (14). If all tasks $T_1,\dots,T_n$ have the same execution time, $s_{copt}(n)$ from Equ. (16) simplifies to s_{opt} from Equ. (7). Depending on the distribution of the task execution times and the values of $P_{dyn}(1)$ and P_{static}, a value smaller than 1 may result for $s_{copt}(n)$ from Equ. (16). In that case, the value has to be rounded up to 1.

If less processors than independent tasks are available, the tasks have to be assigned to the processors in several consecutive rounds. Assuming that p processors are available, each rounds assigns p tasks to the processors for execution. The next section presents an experimental evaluation of the optimal scaling factors.

5 EXPERIMENTAL EVALUATION

The energy model used in this article has been verified by energy measurements on two machines with Intel Sandy Bridge processors (Rauber and Rünger 2012). The measurements have been performed with a complex example application to solve ordinary differential equations with different characteristics. A comparison between measured and predicted energy values shows a deviation that usually lies below 10 %. Thus, the model is suitable for capturing the energy consumption for this class of example applications. To illustrate the quantitative effects of the analytically computed optimal scaling factors, this section describes an experimental evaluation based on randomly generated task sets. In particular, different scaling versions for task executions are compared for different scenarios.

The task sets are generated using different distribution functions (uniform distribution, Gaussian tail distribution, Rayleigh distribution, and Beta distribution), to select the task execution times randomly between 1 sec and 10000 sec. To generate the distributions the GNU Scientific Library has been used. As example, Fig. 2 (right) shows the randomly selected task execution times for a task set of 100 tasks using the different distributions. It can be seen that the Beta distribution tends to select larger task execution times,

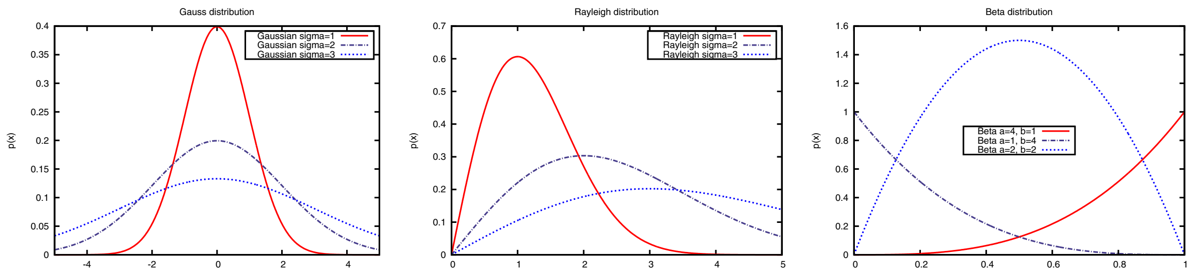


Figure 3: Illustration of the distribution functions used for the selection of the task execution times: Gaussian (left), Rayleigh (middle), Beta (right).

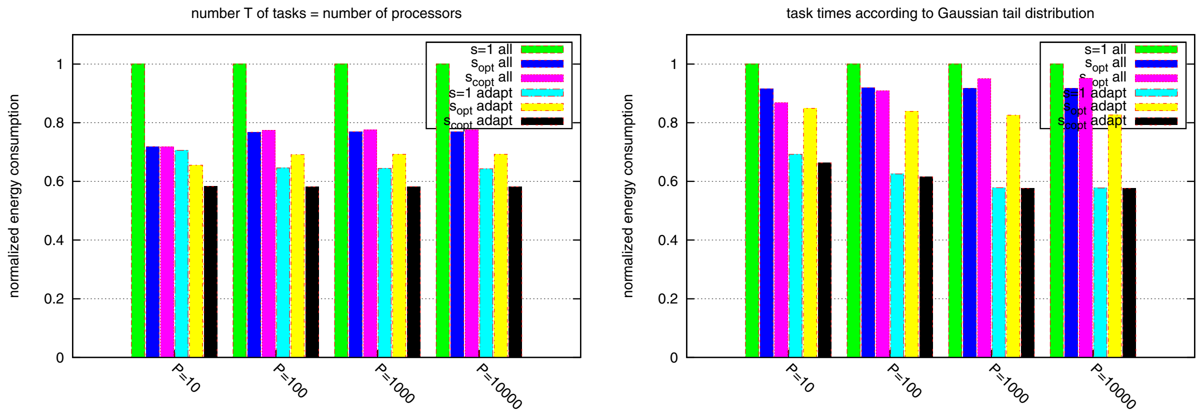


Figure 4: Normalized energy consumption with task execution time created according to a uniform distribution (left) and a Gaussian tail distribution (right). Percentage energy consumption of scaled systems with respect to the unscaled system.

whereas the Gaussian tail distribution favors smaller task execution times. Figure 3 depicts the distribution functions used. The Gaussian tail distribution used results by cutting off the Gaussian distribution at $x = 0$ and considering the resulting right part of the distribution function. For the Gaussian and Rayleigh distribution, the versions with $\sigma = 1$ are used. For the Beta distribution, the version $a = 4$ and $b = 1$ is used.

5.1 Frequency Scaling Factor Variations

Figure 4 (left) compares the percentage energy consumption of scaled systems with respect to the unscaled system. The experiment has been performed for the following numbers of processors: $p = 10$, $p = 100$, $p = 1000$, and $p = 10000$. Each processor is assumed to execute one task. The tasks are generated with randomly selected execution times between 1 and 10000 seconds, leading to a uniform distribution of the task execution times. Each experiment has been repeated 50 times to balance extreme situations caused by the random task creation. The energy consumption has been computed according to Equ. (15) for an example processor with $P_{static} = 4W$ and $P_{dyn}(1) = 20W$. The following six choices of scaling factors for the different processors are compared (from left to right in the diagram): (a) scaling factor $s = 1$ is used for all processors; (b) scaling factor $s = s_{opt}$ according to Equ. (7) is used for all processors; (c) scaling factor $s = s_{copt}$ according to Equ. (16) is used for all processors; (d) $s = 1$ is used for the processor with the largest parallel execution time and Equ. (14) is used to adapt the remaining scaling factors; (e) s_{opt} according to Equ. (7) is used for the processor with the largest parallel execution time and Equ. (14) is

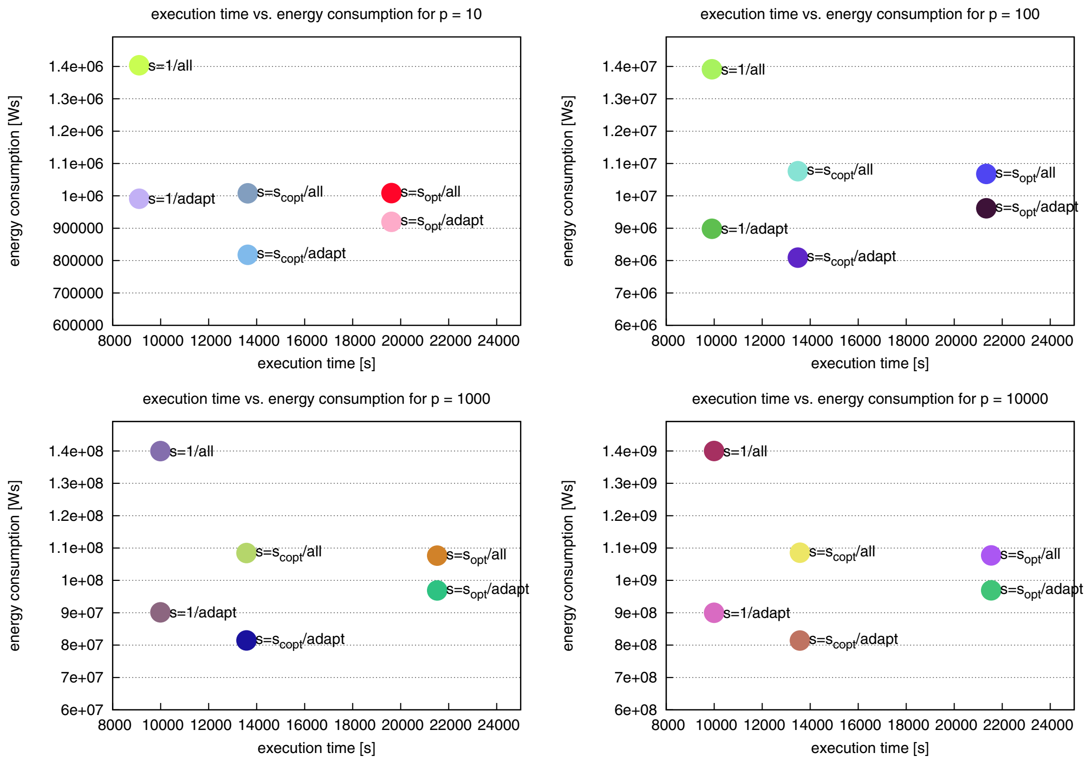


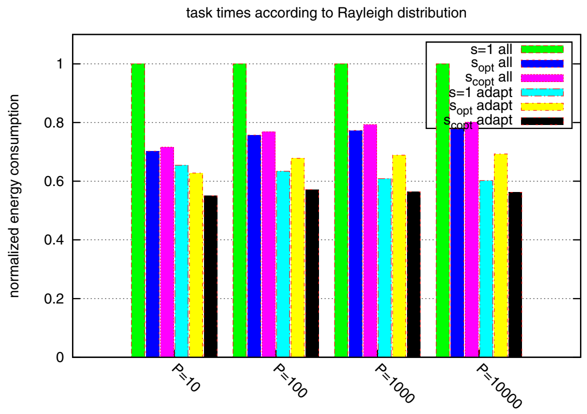
Figure 5: Execution time vs energy consumption for different numbers of processors for a uniform distribution of task execution times.

used to adapt the remaining scaling factors; (f) s_{copt} according to Equ. (16) is used for the processor with the largest parallel execution time and Equ. (14) is used to adapt the remaining scaling factors.

Figure 4 shows that the energy consumption can be considerably reduced by using the frequency scaling factors s_{opt} or s_{copt} instead of $s = 1$. In particular, the scaling factor adaptation can be applied to reduce the total energy consumption significantly. Using s_{copt} for the processor with the largest execution time and adapted scaling factors for the remaining processors leads to the smallest total energy consumption for all numbers of processors. The resulting energy consumption lies below 60% of the energy consumption resulting for the unscaled case, i.e., $s = 1$.

Figure 5 compares the execution times of task sets and the resulting energy consumption for different numbers of processors ($p = 10$, $p = 100$, $p = 1000$, $p = 10000$). 50 sets of randomly generated tasks have been used for the experiments, and the figure shows the average information. Each task in each task set has a randomly generated execution time between 1 and 10000 seconds. For each task set, different scaling factors, the resulting execution times and energy consumptions are computed. The same scaling version as in Fig. 4 are compared. From the figures, the following observations can be made: As expected, using scaling factor 1 always results in the smallest execution time. However, adapting the scaling factors of the other processors can significantly reduce the energy consumption without negatively affecting the overall execution time. Using $s = s_{copt}$ for the processor with the largest execution time leads to a smaller energy consumption than using $s = 1$, but it also increases the resulting execution time accordingly.

Figure 4 (right) compares the choices of frequency scaling factors for a Gaussian tail distribution of task execution times, which has a higher percentage of smaller tasks. For this task distribution, the scaling factor s_{copt} is close to 1, and therefore the adaptive scaling with 1 and s_{copt} lead to similar energy consumptions. Figure 6 (left) uses a Rayleigh distribution for the task execution times. In this case, the non-adapted scaling versions for s_{opt} and s_{copt} already lead to significant energy savings. Figure 6 (right)



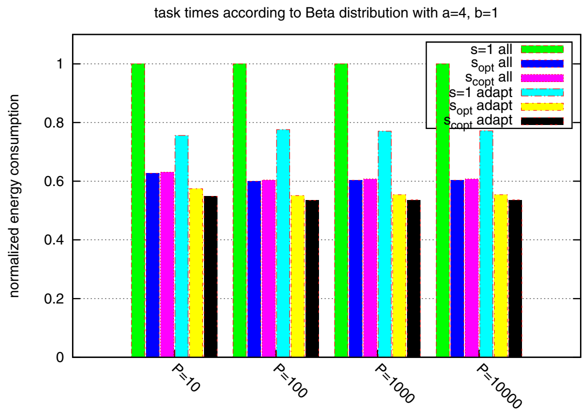


Figure 6: Normalized energy consumption with task execution time created according to a Rayleigh distribution (left) and a Beta distribution (right). Percentage energy consumption.

uses a Beta distribution having a higher percentage of tasks with a larger execution time. In this case, the energy consumption for s_{opt} and s_{copt} in the adapted and non-adapted case are quite similar. Using $s = 1$, the energy consumption is much larger also in the adapted case.

Figure 7 depicts the resulting execution times and energy consumptions for different distributions of the task execution times and different choices of frequency scaling factors for each of these distributions. For $p = 1000$ processors, random task creations according to a uniform, a Gaussian, a Rayleigh, and a Beta distribution are compared. The figure shows that the increase in execution time for the scaled versions compared to the unscaled version strongly depends on the distribution of the task execution time. This increase is small for a Gaussian distribution, tolerable for a uniform and Rayleigh distribution, and large for a Beta distribution. Using the adapted scaling versions always leads to a considerable reduction in energy consumption. The reduction is largest when using $s = 1$ or $s = s_{copt}$ for the largest task and adapting the remaining scaling factors accordingly. The experiments have shown that the effect of choosing frequency scaling factors strongly depends on the distribution of the execution times of the tasks to be executed.

6 RELATED WORK

Power-management features are integrated in computer systems of almost every size and class, from hand-held devices to large servers (Saxe 2010). An important feature is the DVFS technique that trades off performance for power consumption by lowering the operating voltage and frequency if this is possible (Zhuo and Chakrabarti 2008). The approach to determine the voltage scaling factor that minimizes the total CPU energy consumption by taking both the dynamic power and the leakage power into consideration has been discussed in Jejurikar, Pereira, and Gupta (2004), Irani, Shukla, and Gupta (2007), Zhuo and Chakrabarti (2008) for sequential tasks.

The energy consumption of parallel algorithms for shared memory architectures based on the parallel external memory (PEM) model (Arge, Goodrich, Nelson, and Sitchinava 2008) has been discussed in Korthikanti and Agha (2010). In particular, the energy consumption of parallel prefix sums and parallel mergesort is analyzed, but no communication costs are taken into consideration because of the shared memory model. The interaction between the parallel execution and energy consumption is considered in Cho and Melhem (2008) by partitioning a parallel algorithm into sequential and parallel regions and computing optimal frequencies for these regions. No task structuring of the parallel algorithms is considered. Approaches for an energy complexity metric are discussed in Bingham and Greenstreet (2008). Song, Su, Ge, Vishnu, and Cameron (2011) proposes a system-level iso-energy-efficiency model to analyze, evaluate and predict energy-performance of data intensive parallel applications.

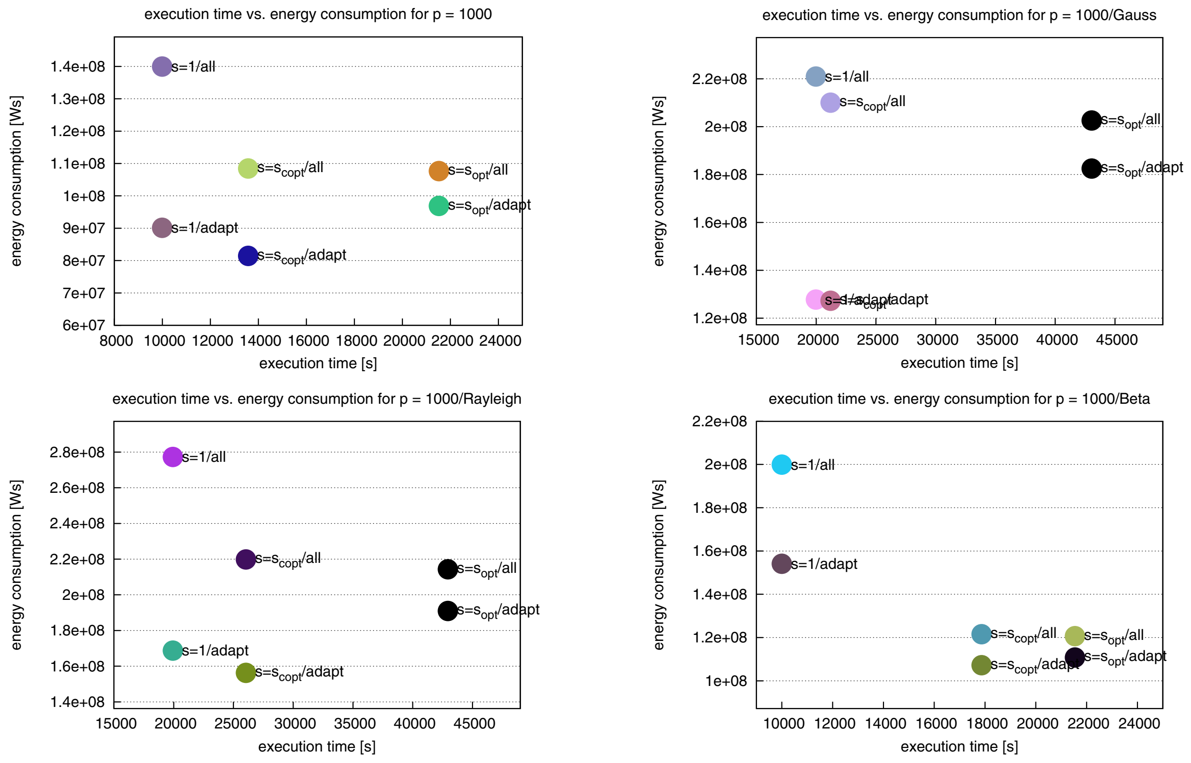


Figure 7: Time vs energy consumption for $p = 1000$ processors using different distributions of task execution times: uniform (top left), Gaussian tail (top right), Rayleigh (bottom left), and Beta (bottom right).

The energy consumption of interconnection networks of chip multiprocessors (CMP) is addressed in Flores, Aragon, and Acacio (2010), Cebrian, Aragon, and Kaxiras (2011). An energy-oriented evaluation of communication optimization for networks is given in Kadayif, Kandemir, Choudhary, and Karakoy (2003) with a focus on sensor networks which have different characteristics as networks in high-performance computing. In the domain of real-time scheduling, many techniques for utilizing available waiting times based on DVFS have been considered, see, e.g., Horvath, Abdelzaher, Skadron, and Liu (2007), Mishra, Rastogi, Zhu, Mossé, and Melhem (2003), Zhu, Melhem, and Mossé (2009). These approaches are usually based on heuristics and are not based on an analytical model as presented in this work.

7 CONCLUSIONS

This article has investigated the energy consumption of the execution of task-based programs using a fork-join based generation of tasks. For concurrently executed tasks, we have analytically derived scaling factors that minimize the overall energy consumption. This investigation is based on hardware frequency scaling techniques. As shown by the analytical model, the scaling factor formula can be used to set the frequency of the processors executing a task such that the energy consumption is reduced by avoiding waiting times at join points. For a specific system, the analytically optimal scaling factors may need to be adjusted to the discrete set of available frequency scaling levels. The result for sequential tasks in a fork-join pattern can be generalized to parallel tasks that employ several processors each. This programming structure is suitable when the number of processors exceeds the number of tasks in a fork-join pattern.

REFERENCES

Arge, L., M. Goodrich, M. Nelson, and N. Sitchinava. 2008. "Fundamental parallel algorithms for private-cache chip multiprocessors". In *SPAA '08: Proc. of the 20th Ann. Symp. on Parallelism in Algorithms and Architectures*, 197–206: ACM.

Bingham, B., and M. Greenstreet. 2008. "Computation with Energy-Time Trade-Offs: Models, Algorithms and Lower-Bounds". In *ISPA '08: Proc. of the 2008 IEEE Int. Symp. on Parallel and Distributed Processing with Applications*, 143–152: IEEE Computer Society.

Butts, J., and G. Sohi. 2000. "A static power model for architects". In *In Proc. of the 33rd Int. Symp. on Microarchitecture (MICRO-33)*, 191–201.

Cebrian, J., J. Aragon, and S. Kaxiras. 2011. "Power-token balancing: Adapting CMPs tp power constraints for parallel multithreaded workloads". In *Proc. of the 25th IEEE Int. Parallel and Distributed Processing Symp. (IPDPS 11)*, 431–442: IEEE.

Cho, S., and R. Melhem. 2008. "Corollaries to Amdahl's Law for Energy". *IEEE Comput. Archit. Lett.* 7 (1): 25–28.

Flores, A., J. L. Aragon, and M. E. Acacio. 2010. "Heterogeneous Interconnects for Energy-Efficient Message Management in CMPs". *IEEE Transactions on Computers* 59:16–28.

Hoffmann, R., and T. Rauber. 2011. "Adaptive Task Pools: Efficiently Balancing Large Number of Tasks on Shared-address Spaces". *International Journal of Parallel Programming* 39 (5): 553–581.

Horvath, T., T. Abdelzaher, K. Skadron, and X. Liu. 2007. "Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control". *IEEE Trans. Comput.* 56 (4): 444–458.

Irani, S., S. Shukla, and R. Gupta. 2007. "Algorithms for power savings". *ACM Trans. Algorithms* 3 (4): 41.

Jejurikar, R., C. Pereira, and R. Gupta. 2004. "Leakage aware dynamic voltage scaling for real-time embedded systems". In *DAC '04: Proceedings of the 41st annual Design Automation Conference*, 275–280: ACM.

Kadayif, I., M. Kandemir, A. Choudhary, and M. Karakoy. 2003. "An energy-oriented Evaulation of Communication Optimizations for Microsensor Networks". In *Proc. of the EuroPar 2003 conference*, 279–286: Springer LNCS 2790.

Kaxiras, S., and M. Martonosi. 2008. *Computer Architecture Techniques for Power-Efficiency*. Morgan & Claypool Publishers.

Korthikanti, V., and G. Agha. 2010. "Towards optimizing energy costs of algorithms for shared memory architectures". In *SPAA '10: Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*, 157–165. New York, NY, USA: ACM.

Lee, Y., and A. Zomaya. 2009. "Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling". In *CCGRID '09: Proc. of the 2009 9th IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, 92–99: IEEE Computer Society.

Mishra, R., N. Rastogi, D. Zhu, D. Mossé, and R. Melhem. 2003. "Energy Aware Scheduling for Distributed Real-Time Systems". In *IPDPS '03: Proc. of the 17th Int. Symp. on Parallel and Distributed Processing*, 21.2: IEEE Computer Society.

N'takpé, T., F. Suter, and H. Casanova. 2007. "A Comparison of Scheduling Approaches for Mixed-Parallel Applications on Heterogeneous Platforms". In *Proc. of the 6th Int. Symp. on Par. and Distrib. Comp.*, 250–257: IEEE.

Rauber, T., and G. Rünger. 2011. "Modeling the Energy Consumption for Concurrent Executions of Parallel Tasks". In *Proc. of the 14th Communications and Networking Simulation Symp. (CNS'11)*, 11–18: SCS.

Rauber, T., and G. Rünger. 2012. "Energy-aware Execution of Fork-Join-based Task Parallelism". In *Proc. of the 20th Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'12)*, 231–240. IEEE.

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

Song, S., C.-Y. Su, R. Ge, A. Vishnu, and K. Cameron. 2011. “Iso-energy-efficiency: An approach to power-constrained parallel computation”. In *Proc. of the 25th IEEE Int. Parallel and Distributed Processing Symp. (IPDPS 11)*, 128–139: IEEE.

Vydyanathan, N., S. Krishnamoorthy, G. Sabin, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz. 2009. “An Integrated Approach to Locality-Conscious Processor Allocation and Scheduling of Mixed-Parallel Applications”. *IEEE Transactions on Parallel and Distributed Systems* 20 (8): 1158–1172.

Zhu, D., R. Melhem, and D. Mossé. 2009. “Energy efficient redundant configurations for real-time parallel reliable servers”. *Real-Time Syst.* 41 (3): 195–221.

Zhuo, J., and C. Chakrabarti. 2008. “Energy-efficient dynamic task scheduling algorithms for DVS systems”. *ACM Trans. Embed. Comput. Syst.* 7 (2): 1–25.