2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)

Towards New Metrics for Appraising Performance and Energy Efficiency of Parallel Scientific Programs

Thomas Rauber
University Bayreuth
Email: rauber@uni-bayreuth.de

Gudula Rünger
Chemnitz University of Technology
Email: ruenger@cs.tu-chemnitz.de

Matthias Stachowski
University Bayreuth
Email: matthias.stachowski@uni-bayreuth.de

***Abstract*—Multiple and conflicting non-functional goals of program execution, such as speedup or energy efficiency, lead to an increasing effort for assessing the program properties. The time-consuming evaluation of program execution is further increased by multiple degrees of freedom influencing the execution behavior, such as the number of threads or the operational frequency of the processors. Furthermore, the efficiency properties strongly depend on the programs or algorithms and may vary when changing the implementation. In this article, we propose new metrics, e.g. the relative power increase factor, which are able to capture the overall quality of program execution such that it is possible to appraise multiple goals depending on different influential factors in an easy-to-use way. The new metrics are evaluated with respect to their usefulness and expressiveness. The investigations are done for the PARSEC benchmark suite on Intel DVFS processors.**

***Keywords*-Energy efficiency, DVFS, Metrics, Multithreading, Performance, PARSEC,**

I. Introduction

The performance and energy efficiency of program execution show a varying behavior depending on a multitude of factors of the software itself and also of the hardware platform used. On the software side, the factors include the algorithm or application chosen and the way how the specific implementation exploits the hardware details of the execution platform. On the hardware side, the factors include the internal organization of the processor architecture as well as the specific setting of hardware parameters chosen for the execution, such as the number of threads employed for execution on multi-cores or the operational frequencies for DVFS processors. However, it is difficult to predict how the parameter setting will influence the performance or energy efficiency of the execution. For example, it is hard to say whether the use of more threads may lead to a smaller execution time of an application, and typically there is an optimal number of threads beyond which the execution time cannot be reduced further. Similarly, the use of a smaller operational frequency may lead to a reduction of the energy consumption, but there is often an optimal operational frequency beyond which a further frequency reduction does not lead to a further energy reduction. Moreover, the ideal case of a linear dependence between execution time and energy consumed, i.e. the lowest energy would be achieved for the fastest program, does not hold due to a power consumption, which can show a large variance for one specific application and more so between different applications.

To support program tuning towards performance and/or energy efficiency, some metrics for evaluating these properties have already been suggested. For the performance evaluation, the traditional speedup, the number of floating point operations per second (flop/s) or the total execution time needed to solve a specific problem (time-to-solution) are used. For evaluating the power or energy efficiency, the metrics include the performance per Watt or the amount of energy needed to solve a problem (energy-to-solution). An attempt to combine the runtime performance and energy consumption in a single fused metrics is the energy-delay product [8].

In this article, we are concerned with the diverse interactions of the effects caused by a varying number of threads and varying frequencies on the performance and the energy consumption of applications, which can be quite intricate. Thus, the question arises whether appropriate metrics can be defined which can capture the dependence of performance and energy consumption depending on the two varying parameter and how an improvement effect can be expressed. In this article, we propose several new metrics, including speedup factors and reduction factors for the execution time and the energy in isolation as well as combined metrics. Also, metrics for the power consumption are introduced with which application-specific power values can be assessed.

The main contribution of this article is to propose metrics that combine the effect of frequency scaling and parallelism for an overall evaluation of multi-threaded application codes on DVFS processors. The new metrics are illustrated and evaluated for the PARSEC benchmarks. Both the Intel Haswell and Skylake architecture are considered. A goal is to provide easy-to-use metrics for a comparison of different applications as well as different execution variants of a single application.

The rest of the article is structured as follows: Section II addresses the power and energy consumption of applications and introduces the basic notation. Section III introduces the hardware and software setting used for the experimental evaluation. Section IV introduces the new metrics and discusses their usefulness based on an experimental evaluation using the PARSEC benchmarks as collection of test examples. Section V discusses related work and Section VI concludes.

978-1-5386-3066-2/17 $31.00 © 2017 IEEE
DOI 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.75

II. Performance and energy consumption

The energy E consumed for the execution of an application code depends on the execution time on the given hardware platform and the power drawing P during the execution. P may vary during the execution of the code. Thus, E is expressed as $E = \int_{t=0}^{t_{end}} P(t)dt$, assuming that the program is executed from time $t = 0$ to time $t = t_{end}$.

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}. Frequency scaling for DVFS processors can be expressed by a dimensionless scaling factor $s \geq 1$ which describes a smaller frequency $f \leq f_{max}$ relative to the maximum possible frequency f_{max} as $f = f_{max}/s$. The maximum possible scaling factor is $s_{max} = f_{max}/f_{min}$. The power drawing P varies with the number of threads p used for the execution and the operational frequency f chosen, so that we can express the power P as a function of p and s, i.e. $P = P(p, s)$. Consequently, the energy consumption also depends on p and s, which can be expressed by

$$E(p, s) = \int_{t=0}^{t_{end}(p,s)} P(p, s)(t)dt. \qquad (1)$$

For simplicity, it is often assumed that $P(p, s)$ is constant during the execution of an application program. In this case, Equ. (1) can be expressed as

$$E(p, s) = P(p, s) \cdot T(p, s) \qquad (2)$$

where $T(p, s) = t_{end}(p, s)$ is the overall parallel execution time, depending on p and s. In practice, E is a discrete function in p and s, since the number of threads is a natural number and the frequency cannot be changed continuously.

Due to the fact that both evaluation criteria for application codes, i.e., execution time and energy consumption, depend on the two independent variables (number of threads, scaled frequency), an evaluation function $\mathcal{E} = (E, T)$ is fully described by a two-dimensional function

$$(E, T) : [1, p_{max}] \times [1 : s_{max}] \subset \mathbb{R}^2 \to \mathbb{R}^2$$

with

$$(p, s) \mapsto (E(p, s), T(p, s)).$$

For each application, these performance functions can be visualized by two surfaces over a 2D grid of (p, s)-points. As example of such an energy surface, the PARSEC application freqmine is shown in Fig. 1. It can be seen that the energy and runtime surfaces show different slopes in both dimensions. When investigating the performance properties of an application, the structure of both surfaces E and T need to be analyzed for either getting a minimum of E or of T or of a combination of both. The notation used in this section is summarized in Table I.

As mentioned above, the power drawing can vary with the number of threads, usually in a way that the power drawing increases with an increasing number of threads. To give an example, Table II shows the power consumption of the PARSEC application freqmine. The table shows that the power drawing increases with the frequency for a fixed number of threads. For a fixed frequency, the power drawing increases up to four threads, corresponding to the number of physical cores, and then it may increase further or may even slightly decrease. Figure 2 shows the power consumption for all PARSEC applications. The upper (lower) diagrams show the power values when the execution is performed with one (eight) thread. It can be observed that the power consumption increases with the operational frequency, as expected, and that there is also an increase of the power consumption with the number of threads used. For one thread, the power values are roughly between 10 and 14 on the Skylake system (Fig. 2 left) and between 15 and 18 Watt on the Haswell system (Fig. 2 right) for the highest frequency of 3.4 GHz. For eight threads, the power values are between 20 and 40 Watt on the Skylake system and between 28 and 52 Watt on the Haswell system for the highest frequency. Different applications change their power consumption behavior in different ways, for example: The benchmark `facesim` has a slightly lower power consumption on eight threads than on one thread with the effect that `facesim` has the highest power consumption on one threads compared to the other applications but has the lowest power consumption on eight threads compared to the other applications. The benchmark `ferret` has a higher power consumption on the Haswell architecture than almost every

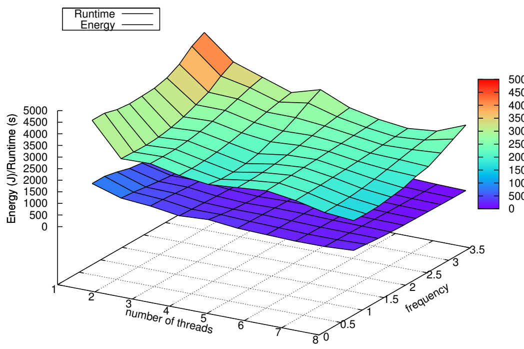


Fig. 1. Energy consumption and execution time of the freqmine benchmark with different frequencies and different numbers of threads on an Intel Core i7 Skylake.

TABLE I
Basic Notation

	Notation	
SYMBOL	UNIT	MEANING
p	scalar	number of threads
f	1/seconds	operational frequency
f_{max}	1/seconds	maximum processor specific frequency
f_{min}	1/seconds	minimum processor specific frequency
$s \geq 1$	dimensionless	scaling factor with $f = f_{max}/s$
s_{max}	dimensionless	maximum scaling factor
$T(p, s)$	seconds	scaled parallel execution time
$P(p, s)$	Watt	power for scaled parallel execution
$E(p, s)$	Joule	energy for parallel execution

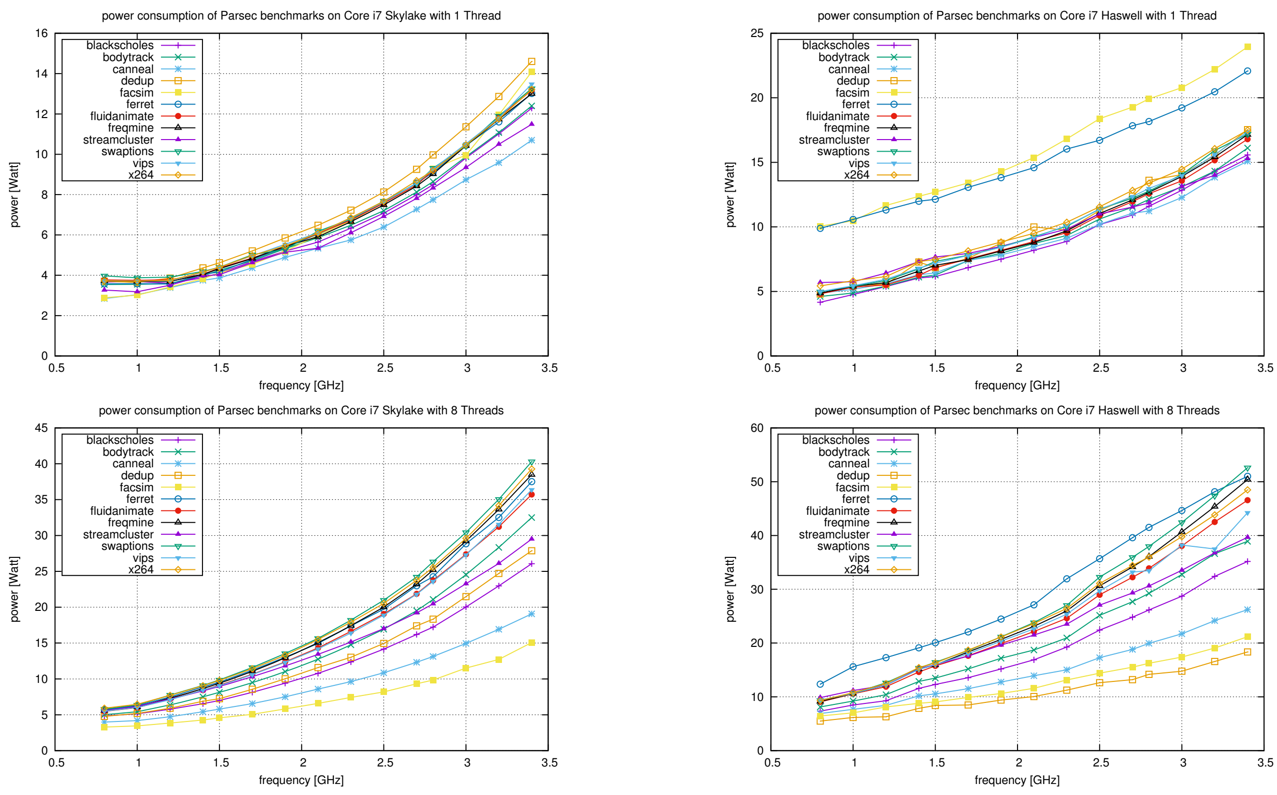


Fig. 2. Power consumption of the PARSEC benchmarks executed with one thread (top) and eight threads (bottom) on an Intel Core i7 Skylake (left) and Haswell (right) architecture using different frequencies.

other benchmark using only one thread. However, the effect weakens for a higher number of threads.

The measurements also show that the power for different applications vary qualitatively and quantitatively in very different ways depending on the number of threads and the frequency chosen. For example for a higher number of threads and a higher operational frequency, the power values show a broader distribution. It can also be observed that there is a larger variation in the power consumption when the number of threads is increased, see Fig. 2.

III. Hardware and Software Settings

The usefulness of the metrics introduced in this article is demonstrated by an experimental evaluation using the Princeton Application Repository for Shared-Memory Computers (PARSEC)[2] Benchmark Suite, which is a collection of benchmarks aiming at the investigation of thread parallelism on recent chip multiprocessors. PARSEC provides programs from a wide range of applications based on the latest techniques in the specific domains and with an emphasis on large workloads as well as multi-core parallelism. Different parallel workloads and execution characteristics are provided. Thus, the PARSEC benchmarks are ideal for demonstrating and testing our metrics proposed. The experiments have been performed with the `native` input set of the PARSEC benchmarks, which is the largest input set available and which is intended for performance measurements on real machines [2]. The energy behavior measured with hardware as well as software techniques has been investigated in [13] with an emphasis on frequency scaling. In this article, we are concerned with the energy and the execution time depending on the two independent variables frequency and number of

TABLE II
Power Consumption in Watt of freqmine on Skylake for different frequencies and different number of threads

	power consumption threads			
Freq	1	2	4	8
0.8 GHz	3.73	3.59	4.56	5.57
1.0 GHz	3.69	3.60	5.71	6.10
1.2 GHz	3.70	4.25	5.91	7.34
1.4 GHz	4.04	4.95	7.96	8.69
1.5 GHz	4.33	5.28	8.48	9.38
1.7 GHz	4.83	5.99	10.21	11.06
1.9 GHz	5.43	6.94	11.94	12.96
2.1 GHz	5.91	8.03	13.78	15.05
2.3 GHz	6.65	9.23	13.51	17.42
2.5 GHz	7.49	10.59	18.40	20.01
2.7 GHz	8.42	12.31	17.72	23.20
2.8 GHz	9.03	13.18	19.20	25.18
3.0 GHz	10.42	15.21	22.20	29.22
3.2 GHz	11.74	11.93	30.95	33.64
3.4 GHz	13.00	13.26	28.63	38.49

TABLE III
PERFORMANCE, POWER AND ENERGY METRICS

SYMBOL	UNIT	MEANING
$S(p,s)$	dimensionless	speedup for scaled case, Eqn.(3)
$R(p,s)$	dimensionless	runtime reduction factor, Eqn.(4)
$EDP(p,s)$	Joule*seconds	energy-delay product EDP, Eqn.(5) parallel and scaled
$ES(p,s)$	dimensionless	energy speedup, Eqn. (6)
$ER(p,s)$	dimensionless	energy reduction factor, Eqn. (7)
$EPS(p,s)$	Joule	energy per speedup, Eqn. (8)
$PS(p,s)$	dimensionless	power speedup, Eqn. (9)
$PI(p,s)$	dimensionless	power increase factor, Eqn. (11)
$RPI(p,s)$	dimensionless	relative power increase factor, Eqn. (12)

threads and especially study the newly defined efficiency characteristics, i.e., energy speedup and runtime speedup, as defined in the next section.

The following experiments have been performed on an Intel Core i7-4770 with the Haswell architecture and an Intel Core i7-6700 with the Skylake architecture, both providing a maximum frequency of 3.4 GHz. The usage of Intel Turbo Boost was disabled on both systems. The processors have four physical cores with hyper-threading, leading to eight logical cores. The memory hierarchy includes an 8 MB shared L3 cache as well as a 256 KB L2 cache and a 32 KB L1 cache per core. The main memory size is 16 GB. The specified thermal design power (TDP) is 84 W on the Haswell system and 65 W on the Skylake system.

The time and energy measurements have been performed using the Running Average Power Limit (RAPL) interface and sensors of the Intel architecture [16], [10]. RAPL sensors can be accessed by control registers, known as Model Specific Registers (MSRs), which are updated in intervals of about 1 ms [10]. In particular, we have used the likwid tool-set, especially the likwid-powermeter in Version 4.0 [18], which provides access to the MSRs introduced above. Experiments have shown that the energy measurement with RAPL sensors are quite accurate when compared to measurements with power-meters [16], [14]. To set the frequencies of the cores to a fixed value, the *cpufreq_set* tool has been used.

IV. Energy and Performance Metrics

In this section, we propose several new metrics, reflecting different performance aspects either depending on p or s. We also include the traditional speedup and energy delay product. Table III summarizes the notation used for the metrics introduced in this section. We distinguish between metrics for performance, energy and mixed considerations.

A. Speedup for scaled execution

The speedup is a major metric used for performance evaluation of parallel application programs. It expresses the relative saving of execution time that can be obtained by using p threads instead of one thread for a fixed frequency [12], and is calculated by dividing the sequential execution time by the parallel execution time. Naturally, the speedup varies when evaluating the same application on different hardware platforms or different application on the same hardware platform.

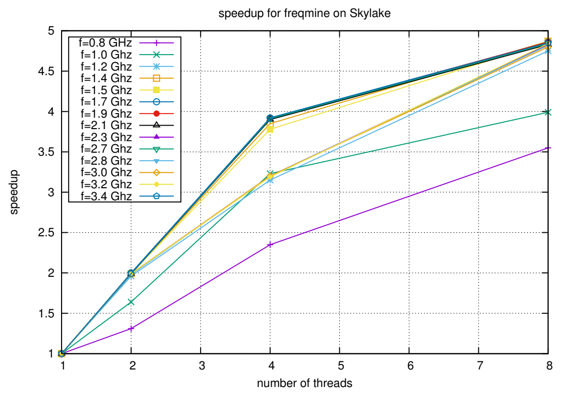


Fig. 3. Speedup for freqmine on Skylake for different frequencies.

An interesting observation for our consideration is that the speedup also varies when evaluation the same application on the same hardware platform but the operational frequency is changed. Thus, we consider the speedup as a metric with two parameters p and s, resulting in a set of speedup values for each number of threads $p \in \{1,\dots,p_{max}\}$ chosen:

$$S(p,s) = \frac{T_{seq}(s)}{T_{par}(p,s)}, \qquad \text{with } s \in \{1,\dots,s_{max}\}. \quad (3)$$

with $T_{seq}(s) = T(1,s)$ and $T_{par}(p,s) = T(p,s)$.

Thus, the speedup of parallel scaled applications is completely depicted as a family of curves with one curve for each frequency chosen. Figure 3 shows the speedup values for the PARSEC benchmark freqmine on Skylake. It is to be mentioned that the problem size is fixed for each application. Although the speedup for scaled parallel programs is a useful und well-accepted metric, there is a need for metrics with a similar flavor, but involving the energy behavior. We propose several possible metrics in the following and discuss they usefulness concerning the appraisal of performance and energy efficiency. Especially, we are interested in finding a metric to evaluate performance and energy efficiency together.

B. Runtime reduction factor

The next metric looks at the performance from a different perspective and captures the relative savings of execution time $T(p,s)$ with respect to $s \in \{1,\dots,s_{max}\}$. This corresponds to the speedup values in Fig 3 when looking at a fixed p, e.g. $p = 4$, and all points on the family of curves that are cut by a vertical line from the x-axis at $p = 4$. As mentioned, the execution time on DVFS processors also varies for varying frequencies and usually increases when the frequency is reduced. To capture how intensively the execution times varies, we define a metric with a similar flavor as the speedup. This is the *runtime reduction factor*:

$$R(p,s) = \frac{T_{par}(p,s)}{T_{par}(p,1)}, \qquad \text{with } p \in \{1,\dots,p_{max}\}, \quad (4)$$

where $T_{par}(p,1)$ corresponds to the highest frequency f_{max} and $T_{par}(p,s)$ corresponds to the smaller frequency $f =$

$f_{max}/s \leq f_{max}$, since $s > 1$. The runtime reduction factor describes the relative increase of the execution time that arises when a smaller operational frequency is used for an arbitrary but fixed number p of threads. Equ. (4) can also be used for the sequential case, using $T_{seq}(s) = T_{par}(1, s)$.

Observation: Figure 4 (left) shows a typical behavior of the runtime reduction factor R, using the PARSEC benchmark freqmine on the Skylake architecture as example. The frequency $f_{max} = 3.4$ represents the unscaled case, i.e., it is $R(p, 1) = 1.0$. In general, $R(p, s) \geq 1$ is expected, since the down-scaled execution time $T_{par}(p, s)$ is usually larger than the un-scaled execution time $T_{par}(p, 1)$. However, the values of $R(p, s)$ do not increase with the same rate as s, which can be seen by the relation $R(p, s) < s$, see Fig. 4 (left).

C. Energy-Delay product EDP

The EDP is defined as the energy consumed by an application program multiplied by its execution time [8], [15]. In analogy to the energy and the parallel execution time, the EDP is a function depending on two variables p and f, merging the two former functions. When including the scaling factor s explicitly, the EDP is expressed as

$$EDP(p, s) = E(p, s) \cdot T(p, s) \qquad [Watt \cdot sec^2] \qquad (5)$$

The EDP combines effects of execution time and energy consumption for different configurations of an application and captures the transformation of energy into useful work.

The significance of the energy delay product can be seen when looking at the energy efficiency. The energy efficiency of an application is defined as performance (flop/sec) per energy unit ($W \cdot sec$) and is measured as $flop/(W \cdot sec^2)$. Given two EDP values EDP_1 and EDP_2 with $EDP_1 < EDP_2$ yields $1/EDP_1 > 1/EDP_2$, both sides measured in $1/(W \cdot sec^2)$, which means that smaller EDP values indicate a better energy efficiency, i.e., a larger performance per energy unit. Therefore, the EDP is a good metric to evaluate the energy efficiency of an application in isolation. However, it is not well suited for comparing different applications, since it delivers the application-specific energy-time product.

D. Energy speedup

Measurements show that the energy consumption for the execution of an application code varies differently for different frequencies and growing numbers of threads. This is caused by variations in the execution time and the power consumption of the individual application codes. Thus, it is reasonable to quantify the changes in the energy consumption $E(p, s)$ by a metric. Analogously to the runtime speedup $S(p, s)$, we define the *energy speedup*

$$ES(p, s) = \frac{E(1, s)}{E(p, s)}, \qquad \text{with } s \in \{1, \ldots, s_{max}\}. \qquad (6)$$

The energy speedup expresses the relative difference of the energy consumption that occurs by using p threads compared to one thread for a fixed frequency, i.e., it captures the energy saving when the number of threads is increased. A minimum value of $ES(p, s)$ for fixed s indicates the value of p for which the energy consumption is at a minimum.

Observation: Investigating the measured energy values, it can be observed that the energy consumption decreases when using a larger number of threads. Therefore, it is expected that $ES(p, s) > 1$. This is confirmed for most of the PARSEC applications, as can be seen in Column 5 of Table IV. There are a few exceptions, which are caused by applications with a very small energy consumption. For most applications, the energy speedup is larger for smaller frequencies compared to larger frequencies.

E. Energy Reduction Factor

When considering a varying frequency and its effect on the energy consumption, we propose the *energy reduction factor*:

$$ER(p, s) = \frac{E(p, s)}{E(p, 1)}, \qquad \text{with } p \in \{1, \ldots, p_{max}\}\ . \qquad (7)$$

The energy reduction factor expresses the relative difference of the energy consumption when using a smaller frequency $f = f_{max}/s$ instead of f_{max} for a fixed number of threads. Thus, ER is similar to ES with the difference that varying frequencies are compared and the number of threads is arbitrary but fixed. For $s = 1$, the ER value is 1, i.e., $ER(p, 1) = 1$. For larger s, an ER value smaller than 1, i.e., $ER(p, s) < 1$, results, if less energy is consumed than for the maximum frequency 3.4 GHz. But also an ER value larger than 1 can occur, if this is not the case. For a given application and a fixed p, a minimum value of $ER(p, s)$ indicates the frequency at which the energy consumption is at a minimum.

Observation: Table IV shows ER values for all PARSEC benchmarks in columns 7 and 8. More precisely, column 7 considers $ER(1, s)$ and gives the minimum ER value for all $s \geq 1$. Analogeously, column 8 considers $ER(8, s)$ and and gives the minimum ER value for all $s \geq 1$. In all cases, these minimum ER values are smaller than 1. This indicates that there exists a frequency for which energy can be saved, since smaller ER values correspond to more energy saving. The energy measurements show that the minimum energy consumption typically occurs for frequencies between 2.0 GHz and 2.5 GHz on both Haswell and Skylake, which means that it is not the case that the smallest possible frequency of 0.8 GHz leads to the smallest energy consumption.

F. Energy per Speedup

A metric to set the energy consumption in relation to the (runtime) speedup of a specific application program is the metric *energy per speedup*, which we define as

$$EPS(p, s) = \frac{E(p, s)}{S(p, s)} \qquad [J]. \qquad (8)$$

The energy per speedup captures the amount of energy that must be invested for each speedup unit achieved. Small values of EPS indicate an efficient use of energy for parallel programs when using more threads to obtain a faster execution.

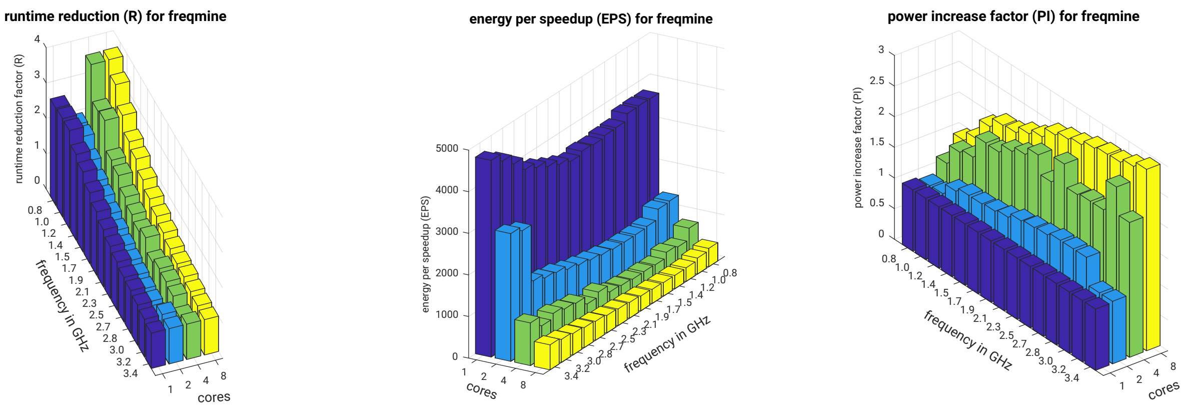


Fig. 4. Runtime reduction factor R (left), energy per speedup EPS (middle), and power increase factor PI (right) for the PARSEC application freqmine on Skylake.

Interestingly, the EPS is strongly related to the EDP from Equ. (5) due to:

$$EPS(p,s) = \frac{EDP(p,s)}{T_{seq}(s)},$$

i.e., the EPS normalizes the EDP with respect to the application-specific sequential runtime for the same frequency.

Observation: Figure 4 (middle) shows the EPS values for the freqmine benchmark on Skylake. It can be seen that the EPS values vary with the operational frequency in an u-shape manner. For different frequencies, different amounts of energy are required for each unit of the runtime speedup, and the smallest amount of energy is required for medium frequencies around 2.3 GHz. For the freqmine application, the EPS values decrease with increasing p, i.e., using more threads is beneficial for the energy consumption. This is a typical behavior that can be observed for most PARSEC applications as long as they show a good performance scalability.

G. Power Speedup

A speedup metric for the power is derived from the energy speedup and the runtime speedup. The *power speedup* $PS(p,s)$ is defined by using the (runtime) speedup and the energy speedup in the following way:

$$PS(p,s) = \frac{ES(p,s)}{S(p,s)} \tag{9}$$

By re-formulating this formula, the power speedup reduces to

$$\begin{aligned} PS(p,s) \quad &= \tfrac{E(1,s)}{E(p,s)} \cdot \tfrac{T_{par}(p,s)}{T_{seq}(s)} \\ &= \tfrac{E(1,s)}{T_{seq}(s)} \cdot \left(\tfrac{E(p,s)}{T_{par}(p,s)}\right)^{-1} \\ &= \tfrac{P(1,s)}{P(p,s)}, \end{aligned} \tag{10}$$

thus only using the power itself and not the time. As the calculation shows, $PS(p,s)$ is a dimensionless metric which captures the relative difference of the power consumption of an application program that occurs when using p threads instead of a sequential execution for an arbitrary but fixed frequency. Since the power consumption increases with the number of threads for most applications, see Fig. 2, the expected value for the power speedup is smaller than 1.

Observation: Figure 5 shows the power speedup for the Skylake processor for all PARSEC benchmarks using selected operational frequencies ($f = 0.8$ GHz, $f = 2.1$ GHz, $f = 3.4$ GHz). Similar data have been measured for the Haswell processor. The figure shows that the PS values decrease with the number of threads for most applications. Exceptions are the dedup, facesim, and ferret applications, which show an irregular behavior due to their very small overall energy consumption and execution time. The figure also shows that the power speedup usually decreases with increasing operational frequencies. The power speedup can also be used to compare different applications concerning their power consumption behavior: Considering two applications A_1 and A_2 with $PS_{A_1}(p,s) > PS_{A_2}(p,s)$, this property expresses that A_1 shows a smaller power increase than A_2 for an increasing number of threads and the same operational frequency, i.e., for this frequency the power sensitivity of A_1 is smaller than the power sensitivity of A_2.

H. Power Increase Factor

For a joint evaluation of execution time and energy consumption, the following metric is well suited. Considering the power consumption of an application program using the same operational frequency corresponding to scaling factor s, it can be observed that the power typically increases with the number of threads employed, see Table II. This effect can be quantitatively captured with the *power increase factor* $PI(p,s)$, which we define as follows:

$$PI(p,s) = \frac{P(p,s)}{P(1,s)}, \qquad s \in \{1,\ldots,s_{max}\}\ , \tag{11}$$

The metric PI is in fact the inverse of the power speedup PS.

Observation: Figure 4 (right) shows the power increase factor PI for the freqmine application on Skylake as an example, showing that $PI(p,s)$ is not a linear function in p, which means that the power increase is not proportional to the number of threads employed. However, it can be seen

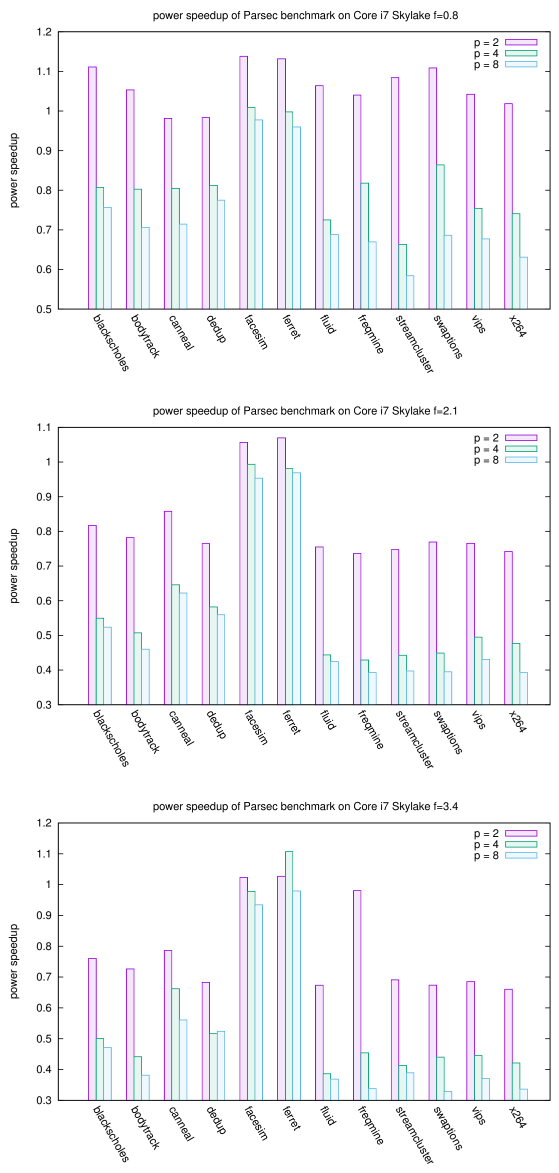


Fig. 5. Power speedups for selected frequencies on an i7 Skylake: 0.8 GHz (top), 2.1 GHz (middle), and 3.4 GHz (bottom).

that the PI values increase with p for a fixed f. Comparing different frequencies, some variations occur with a tendency towards larger PI values for larger frequencies.

A comparison between PI and the (runtime) speedup S shows that there is a direct correlation between the runtime speedup and the power increase. Thus, the runtime speedup seems to be a good indicator for the effect, whether for a given application a significant increase of the power consumption can be expected for a larger number of threads. This observation gives rise to the next metric.

I. Relative Power Increase Factor

To further investigate the correlation between runtime speedup and power increase, we define the *relative power increase factor* as a quantitative measure for the mutual dependence of the runtime speedup and the power increase factor as follows:

$$RPI(p,s) = \frac{PI(p,s)}{S(p,s)} \tag{12}$$

$RPI(p,s)$ describes how much the power consumption of an application increases per runtime speedup if the number of threads is increased accordingly. Small RPI values for an application indicate a good power usage, i.e., the power increase is small compared to the performance increase when increasing the number of executing threads. If there were a linear dependence of the power consumption on the runtime speedup, we would expect an RPI value of about 1. This is indeed the case for some of the PARSEC benchmarks, but for many benchmarks, it is $RPI < 1$, i.e. the power increase is smaller than the runtime speedup.

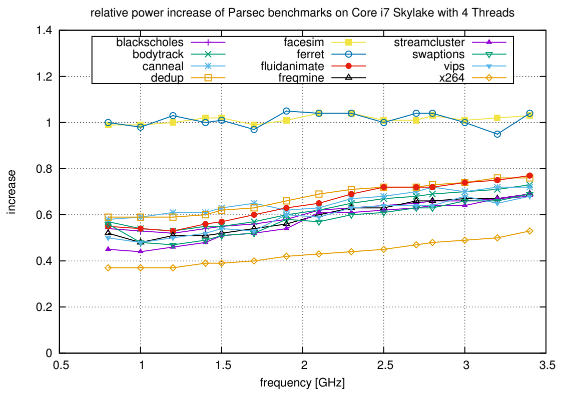
$RPI(p,s)$ is a metric that captures effects of both independent variables p and s, and is thus able to illustrate the influence of both. Also, both the energy consumption and the parallel execution time are evaluated together. Thus, this metric is capable to reduce the influence of two independent variables on the two dependent phenomena to one metric.

Observation: Figure 6 shows the RPI values of all PARSEC applications for varying frequencies. For the illustration, $p = 8$ has been chosen, since the processors used have eight logical cores. It can be seen that the RPI values are below 1.0 for most applications. Thus, for the energy consumption of these applications, it is useful to employ more threads as long as the RPI value decreases with an increasing number of threads. When comparing different applications, smaller RPI values indicate an advantage for a parallel execution, since applications with smaller RPI values exhibit a better power usage than applications with larger RPI values.

Figure 6 also shows that RPI has typically values between 0.5 and 1, so the metric is easy to read and interprete compared to other metrics such as the EDP. Furthermore, the RPI metric is suitable to study and compare different applications or implementations concerning the correlation between power or energy efficiency on the one hand and effective parallelism on the other hand. A coarse evaluation criteria is that a lower RPI-value means that the inherent parallelism of an application is exploited without a significant increase of the power drawing, thus leading to a smaller overall energy consumption.

J. Evaluation summary

Table IV summarizes the values of the metrics introduced for the PARSEC benchmarks on both the Haswell (top) and the Skylake (bottom) architecture. The table shows for each PARSEC application (a) the minimum and maximum energy consumption in Joule, taken over all possible frequencies and number of threads (columns 1 and 2), (b) the speedup obtained when using $p = 8$ threads for frequencies $f = 0.8$ GHz



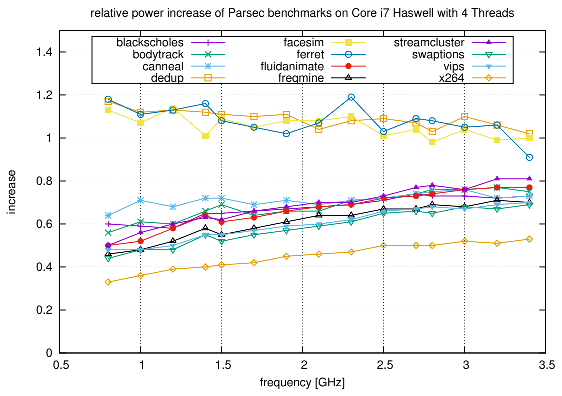


Fig. 6. Relative Power Increase for different frequencies using four threads on an i7 Skylake (left) and Haswell (right)

and $f = 3.4$ GHz (columns 3 and 4) (c) the power increase factor PI for frequencies $f = 0.8$ GHz and $f = 3.4$ GHz (columns 5 and 6) (d) the smallest energy reduction factor over all frequencies for $p = 1$ and $p = 8$ (columns 7 and 8) (e) the energy speedup ES for $p = 8$ threads (columns 9 and 10) (f) the minimum and maximum energy per speedup EPS in Joule, taken over all possible frequencies and number of threads (columns 11 and 12), and (g) the minimum and maximum relative power increase factor RPI, taken over all frequencies and number of threads (columns 13 and 14).

Comparing the Haswell and Skylake architectures, it can be observed that the Skylake processor needs significantly less energy for executing an application than the Haswell processor, indicating a more energy-efficient architectural design. For the energy speedup, large variations can be observed for the different applications using different frequencies, and there is no clear distinction between Haswell and Skylake concerning the energy speedup. The RPI values, which are shown in the last columns, show a surprising match of the values on the two processors, whereas the energy values and the execution times are quite different. This indicates that the RPI expresses the essence of the application's performance behavior in an architecture-independent way. Only the applications with a very low execution time (such as dedup, facesim, ferret from PARSEC) have an RPI larger than 1. The other applications typically have RPI values between 0.5 and 1, with the application x264 as an exception with a minimum RPI value of 0.29 and 0.32, respectively. However, x264 is the application which achieves the highest speedup, thus supporting the observation of the correlation between the runtime speedup and power efficiency expressed by the RPI value.

V. Related Work

Power-management techniques and features have been integrated in computer systems of different sizes and classes, from handheld devices to large servers [17], [4], [5]. An important feature for energy saving is the DVFS technique that trades off performance for power consumption by lowering the operating voltage and frequency if this is possible [19].

The energy delay product (EDP) has first been introduced in [8], [7] to capture both energy consumption and execution time simultaneously, see also [3]. However, as it has been pointed out in [1], the EDP is not able to distinguish whether a smaller execution time or a better energy utilization leads to a smaller EDP value when comparing different version of a program. The biased effects of the EDP and its generalization ED^n for a nonnegative integer n are also discussed in [9]. To separate energy and performance when considering different program versions, [1] introduces powerup and greenup metrics, which are similar to our power speedup and energy speedup metrics. These metrics are then used together with the speedup to identify different energy categories of software. In our article, we go into a different direction by investigating the relationship between speedup, energy, and power. In the context of frequency scaling, [6] defines the power-aware speedup, which incorporates the frequency when computing the runtime speedup of programs. However, the energy consumption is not explicitly taken into consideration. In the context of task scheduling, [11] has proposed the metrics speedup per Watt (SPW), power per speedup (PPS), and energy per target (EPT).

VI. Conclusions

Performance and energy consumption are two important characteristics of application codes, especially if a parallel execution is considered. Today's goals for application codes are maximum performance and minimum energy consumption. However, the relation between performance and energy are quite complex so that metrics are needed to assess the quality of the code concerning these non-functional properties. We have proposed and evaluated several metrics including energy speedup, energy reduction factor, energy per speedup, power speedup, power increase factor, and relative power increase factor, which capture the effects of a varying number of threads and varying frequencies on performance and energy efficiency in isolation or in a combined measure.

Especially the novel metric relative power increase factor RPI is capable to capture all effects and integrate them into a single value. The metric RPI has also the advantage that due to its scale between about 0.5 and 1, the application-specific

TABLE IV
EVALUATION OF THE METRICS FOR THE PARSEC BENCHMARKS ON HASWELL (FIRST HALF) AND SKYLAKE (SECOND HALF)

	Energy Consumed		**Speedup (p=8)**		**PI (p=8)**		**ER (min)**		**ES (p=8)**		**EPS**		**RPI (p=8)**	
Benchmark	min	max	f=0.8	f=3.4	f=0.8	f=3.4	p=1	p=8	f=0.8	f=3.4	min	max	min	max
blackscholes	948.56	2323.36	3.60	3.54	1.77	2.26	0.74	0.61	2.04	1.57	260.22	2323.36	0.49	0.64
bodytrack	868.69	2134.53	3.72	3.47	1.76	2.42	0.76	0.56	2.11	1.44	234.26	2134.53	0.47	0.71
canneal	1209.09	2851.98	2.59	2.66	1.42	1.74	0.61	0.50	1.83	1.53	460.53	2851.98	0.55	0.67
dedup	3.72	5.91	1.00	1.00	1.14	1.05	0.71	0.57	0.87	0.95	3.72	5.83	1.02	1.15
facesim	3.84	6.07	1.00	0.93	1.14	0.97	0.72	0.69	0.88	0.95	3.84	6.12	0.99	1.18
ferret	3.63	5.89	1.00	0.94	1.16	0.88	0.66	0.65	0.86	1.07	3.63	6.00	0.93	1.17
fluidanimate	1949.83	5077.73	4.32	4.02	1.88	2.77	0.72	0.53	2.30	1.45	453.43	5077.73	0.44	0.69
freqmine	2742.39	7700.87	4.67	4.65	1.88	2.94	0.73	0.53	2.49	1.58	586.20	7700.87	0.40	0.63
streamcluster	1564.56	5222.48	5.19	4.99	1.73	2.59	0.65	0.42	3.00	1.93	315.75	5442.64	0.33	0.52
swaptions	1411.24	4313.39	5.18	5.18	1.88	3.06	0.71	0.54	2.76	1.69	269.41	4313.39	0.36	0.59
vips	648.34	1703.58	4.30	3.76	1.88	2.55	0.73	0.54	2.28	1.48	151.61	1703.58	0.44	0.68
x264	560.11	2155.81	5.95	5.75	1.71	2.77	0.75	0.53	3.48	2.08	94.92	2155.81	0.29	0.48
blackscholes	539.53	1546.02	2.97	3.70	1.32	2.12	0.84	0.75	2.25	1.74	147.55	1546.02	0.44	0.57
bodytrack	490.99	1314.92	2.91	3.91	1.42	2.62	0.85	0.71	2.06	1.49	126.06	1314.92	0.46	0.67
canneal	650.01	2015.27	2.80	2.77	1.40	1.78	0.72	0.65	2.00	1.55	232.10	2015.27	0.50	0.64
dedup	84.86	207.75	2.36	2.65	1.29	1.91	0.84	0.87	1.83	1.39	30.00	207.75	0.54	0.79
facesim	2.33	3.47	1.01	1.00	1.02	1.07	0.79	0.83	0.98	0.93	2.32	3.75	1.01	1.07
ferret	4.53	7.17	1.03	0.99	1.04	1.02	0.73	0.91	0.99	0.97	4.09	8.28	0.98	1.07
fluidanimate	1151.72	3151.76	2.97	3.95	1.45	2.71	0.83	0.70	2.04	1.46	282.69	3151.76	0.48	0.69
freqmine	1543.42	4726.40	3.55	4.83	1.49	2.96	0.82	0.67	2.38	1.63	322.56	4726.40	0.41	0.61
streamcluster	878.56	3851.71	5.29	4.77	1.71	2.57	0.75	0.55	3.09	1.86	144.19	3851.71	0.32	0.54
swaptions	779.57	2529.71	3.68	5.31	1.46	3.04	0.86	0.68	2.53	1.75	149.48	2529.71	0.39	0.57
vips	367.38	1067.31	3.28	4.27	1.48	2.70	0.84	0.68	2.22	1.58	84.57	1067.31	0.44	0.63
x264	304.57	1197.61	4.92	6.20	1.58	2.97	0.87	0.69	3.11	2.09	48.37	1197.61	0.32	0.48

values are easy to grasp and interprete by the application programmer. Not only the absolute values but also significant deviations or anomalies are quickly detected. The investigation for the PARSEC benchmark also indicates that the RPI captures the essence of an algorithm in an hardware-independent way. Thus, the RPI metric is a promising and easy-to-use metric for a combined assessment of energy consumption and performance depending on both variables.

ACKNOWLEDGMENT

This work was partially performed within the Federal Cluster of Excellence EXC 1075 MERGE Technologies for Multifunctional Lightweight Structures and supported by the German Research Foundation (DFG). This work is also supported by the German Ministry of Science and Education (BMBF), project SeASiTe, number 01IH16012A/B.

REFERENCES

[1] S. Abdulsalam, Z. Zong, Q. Gu, and Meikang Qiu. Using the Greenup, Powerup, and Speedup metrics to evaluate software energy efficiency. In *Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*, pages 1–8, Dec 2015.

[2] C. Bienia, S. Kumar, J.P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. of the 17th Int. Conf. on Parallel Architectures and Compilation Techniques*, October 2008.

[3] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoğlu, J. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE-MICRO*, 20(6):26–44, 2000.

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

[6] R. Ge and K. W. Cameron. Power-Aware Speedup. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, March 2007.

[7] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, Sep 1996.

[8] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Proceedings of 1994 IEEE Symposium on Low Power Electronics*, pages 8–11, Oct 1994.

[9] C. H. Hsu, W. C. Feng, and J. S. Archuleta. Towards efficient supercomputing: a quest for the right metric. In *19th IEEE Int. Parallel and Distributed Proc. Symp.*, pages 8 pp.–, April 2005.

[10] Intel. *Intel 64 and IA-32 Architecture Software Developer's Manual, System Programming Guide*, 2011.

[11] J. Mair, K. Leung, and Z. Huang. Metrics and task scheduling policies for energy saving in multicore computers. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 266–273, Oct 2010.

[12] T. Rauber and G. Rünger. *Parallel Programming for Multicore and Cluster Systems*. Springer Verlag, 2013.

[13] 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.

[14] 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.

[15] G. Rong, F. Xizhou, S. Shuaiwen, C. Hung-Ching, L. Dong, and K.W. Cameron. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658 –671, may 2010.

[16] E. Rotem, A. Naveh, A. Ananthakrishnan, D. Rajwan, and E. Weissmann. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro*, 32(2):20–27, 2012.

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

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

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