Modeling the Energy Consumption for Concurrent Executions of Parallel Tasks

Thomas Rauber
University Bayreuth
rauber@uni–bayreuth.de

Gudula Rünger
Chemnitz University of Technology
ruenger@informatik.tu–chemnitz.de

Keywords: energy model, task-based programming, communication;

Abstract

Programming models using parallel tasks provide portable performance and scalability for modular applications on many high-performance systems. This is achieved by the flexibility of a two-level programming structure supporting mixed task and data parallelism. Due to the emerging importance of energy efficiency in high-performance computing, programming models with parallel tasks should be extended to be able to include energy concerns. Based on a well-accepted analytical energy model for a processor's energy consumption, this article explores the energy consumption of parallel tasks with communication that are executed concurrently with other tasks. Simulations show the different energy consumption scenarios for different task cooperations and demonstrate the potential for a flexible energy usage on varying parallel platforms.

1. INTRODUCTION

Today, power management of hardware systems and the ability to reduce power consumption of underutilized system resources are offered by computer systems of almost every size and class. As discussed widely in the literature, a useful power management also requires the software to be designed appropriately such that the hardware features can be exploited, see e.g. [17] using the term power-efficient software. Efficiency of software addresses the proportionality of resource utilization versus useful work done.

For parallel computations, the computational efficiency has long been the major goal. The shift to a power efficient parallel computation requires the development of an efficiency terminology and energy models in high performance computing. Also, flexible software is needed that can adapt to the hardware situation with the goal to minimize energy consumption while all the work required is performed.

This article considers the parallel programming model of parallel tasks, also called parallel modules, multiprocessor tasks or malleable tasks [20, 19, 14, 15, 16], which has been proven useful for structuring modular application problems. Using this model, the specification of the coarse grain parallelism expressed by parallel tasks is separated from the actual implementation and execution. A well-structured development process can be used to transform the task-based specification into an efficient parallel implementation; this can be done using a library like Tlib [16] or by compilation support, see, e.g., the TwoL compiler [15].

A key step in the transformation process is to determine the computation order of independent tasks and the assignment of execution cores to each individual task. To support the assignment, a variety of heuristics have been proposed. The basis of these algorithms is a cost model, which can be simple, like the PRAM model, or can be more sophisticated based on measured data or a performance prediction model [11, 9]. The goal of a cost model is to get a good plan how the parallel tasks are placed onto the system resources. An exact performance prediction is usually not needed. Most important is the flexibility of the programming model, which needs one specification of an application program only and can then set the parallelism within each task and between tasks in an adaptive way so that on each hardware platform an individual software leads to low computation time and high efficiency.

In this article, we propose to exploit this programming model for power-efficient software. The contribution of this article is to bring together the parallel programming model of parallel tasks and well-known energy consumption models. Based on the energy consumption model for single cores, an analytical energy model for concurrently executed parallel tasks is proposed. With such a model, the flexibility of the programming model can be exploited to develop individual energy-efficient programs for specific hardware systems from the same software specification.

The rest of the article is organized as follows: Section 2. summarizes the task parallel programming model. Section 3. discusses related work. Section 4. presents the energy model and Section 5. uses this model to explore the energy consumption of concurrent parallel tasks. Section 6. considers the energy-consumption for task executions and derives optimal frequency scaling factors. Section 7. concludes the paper.

2. TASK PARALLEL PROGRAMMING

The programming model using parallel tasks is a two-level programming model which distinguishes between a coarse-grain upper level consisting of cooperating parallel tasks and a fine-grain lower level capturing parallelism within those tasks. Using this programming model, the application to be implemented is decomposed into well-defined modules in a top-down fashion. Each module itself is implemented as a parallel function, called parallel task, to be executed on a

variable number of execution cores, called processors in the following; this is the fine-grain parallelism. The cooperation and coordination between the tasks is the coarse-grain level of parallelism.

The specification of the coarse-grain module structure and the fine-grain internal parallelism of modules represents a correct parallel structure of the application and is the basis for a variety of different parallel implementations. These implementations vary in the way in which the parallelism provided is exploited at the module level as well as within each module, caused by different numbers of processors used for the individual modules. A good choice leading to high efficiency and scalability can be planned based on a cost model representing the parallel execution time of individual modules as well as the module program as a whole. Costs express the parallel execution time including communication time in a flexible way, reflecting the dependence of these costs on the degree of parallelism exploited and the execution order of the tasks.

This section describes the programming model using parallel tasks in more detail with an emphasis on the cost model and illustrates the basic implementation decisions for an actual execution platform. The subsequent section will then describe how this programming model can be extended to include energy concerns.

2.1. Task specification

A task parallel program consists of a set $\mathcal{T}$ of parallel tasks. Each parallel task $T \in \mathcal{T}$ is a piece of parallel code with a set of input parameters I and a set of output parameters O. These parameters may include scalar values as well as structured data structures, such as arrays. $I \cap O \neq \emptyset$ is possible, since input parameters may be used or modified resulting in output data. An entire parallel program is built up of the tasks $T \in \mathcal{T}$ by combining cooperating tasks in one of the following ways:

(1) A *consecutive cooperation* of task T_1 with input I_1 and output O_1 and task T_2 with input I_2 and output O_2 is used when $I_2 \cap O_1 \neq \emptyset$, i.e., task T_2 needs input data computed by T_1. A consecutive cooperation is denoted as $T_1 \circ T_2$.

(2) A *parallel cooperation* of task T_1 with input I_1 and output O_1 and task T_2 with input I_2 and output O_2 can be used if there are no control dependencies and no data dependencies between T_1 and T_2, i.e., $I_1 \cap O_2 = \emptyset$ and $I_2 \cap O_1 = \emptyset$. A parallel cooperation is denoted as $T_1 || T_2$.

(3) A *hierarchical structure* can be defined if a task $T \in \mathcal{T}$ has an internal structure suitable for task parallelism, i.e., task T is built up of other tasks $T_1, \ldots, T_k \in \mathcal{T}$, which cooperate in a parallel or consecutive way.

Both a consecutive and a parallel cooperation of tasks can be generalized to an arbitrary number of tasks and the resulting tasks can be used for further cooperation with other tasks.

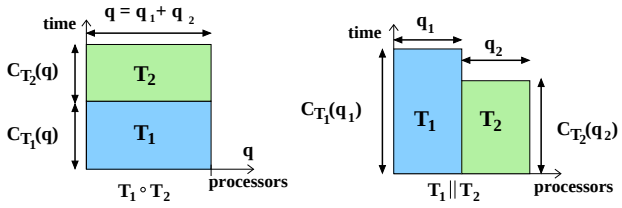

Figure 1. : Illustration of consecutive execution $T_1 \circ T_2$ (left) and parallel execution $T_1 \parallel T_2$ (right) of two tasks. For the consecutive execution, all q processors are partitioned in two disjoint sets of size q_1 and q_2, respectively, with $q_1 + q_2 = q$ and q_i processors are used for executing T_i, $i = 1, 2$.

2.2. Execution times for parallel tasks

The overall parallel execution time of task parallel programs depends on the execution time of the parallel tasks $T \in \mathcal{T}$ and the actual allocation of processors for the execution of these tasks. Given a set P of processors with $|P| = p$, the overall parallel execution time also depends on p and the subdivision of P into subsets where each subset executes individual tasks one after another. The parallel execution time of task $T \in \mathcal{T}$ is given as a function C_T that depends on the number of processors q used for the computation of T and producing a real number representing the costs, i.e.,

$$C_T(q) : \{1, \ldots, p\} \rightarrow \mathbb{R} \quad .$$

Such a function can express the actual execution time on a specific hardware platform, which can be a measured time in seconds or a predicted time. Also, more simplified 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 be assumed to be fixed.

The parallel machine considered consists of a set of homogeneous processors connected by an interconnection network for message passing communication. For such parallel machines, the actual parallel execution time is usually a non-linear function caused by the behavior of collective communication operations. Thus, we assume that $C_T(q)$ is a non-linear function in q.

The execution time for the entire parallel program consisting of task set $\mathcal{T}$ is built up from the functions $C_T(q), T \in \mathcal{T}, 1 \leq q \leq p$, according to the structure of the tasks:

(1) If tasks T_1 and T_2 are executed consecutively on the same set of q processors, the execution time is:

$$C_{T_1 \circ T_2} = C_{T_1}(q) + C_{T_2}(q),$$

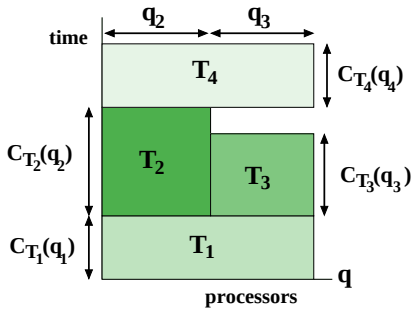

Figure 2. : Illustration for the execution of a hierarchically structured task $T \equiv T_1 \circ (T_2 \parallel T_3) \circ T_4$ on q processors. The execution of T_2 and T_3 is performed concurrently on q_2 and q_3 processors with $q_2 + q_3 = q$ after the execution of T_1 on all $q_1 = q$ processors and before the execution of T_4 on all $q_4 = q$ processors.

see Fig. 1 (left) for an illustration of the costs in a Gantt diagram.

(2) If tasks T_1 and T_2 are executed concurrently on disjoint subsets of processors of size q_1 and q_2, respectively, with $1 \leq q_1, q_2 \leq p$, $2 \leq q_1 + q_2 \leq p$, the execution time is:

$$C_{T_1 || T_2} = \max_{i=1,2} \{C_{T_i}(q_i)\}$$

assuming that the execution of T_1 and T_2 start at the same time, see Fig. 1 (right) for an illustration.

(3) The hierarchical execution form expresses an internal task structure and, thus, the execution time of a task T built up of $T_1, \ldots, T_k \in \mathcal{T}$ is captured by the execution times of $T_1, \ldots, T_k$ combined according to the internal structure of T, i.e.,

$$C_T(q) = \text{ combine } (C_{T_1}(q_1), \ldots, c_{T_k}(q_k))$$

where the combine-function captures a structural composition due to the cases (1) and (2), see Fig. 2.

For one specification of a task parallel program, there are several possibilities to actually execute the program according to specific implementation decisions. These decisions include the choice of a specific number of processors q_T for each of the tasks $T \in \mathcal{T}$ and the decision on the execution order in case (2). For the actual execution, this lack of dependencies in case (2) can be exploited by a parallel execution with execution time functions as given above in case (2) or, alternatively, by a consecutive execution of T_1 and T_2 with a different function similar to case (1). This alternative might be advantageous if not enough processors are available or a higher number of processors is needed for each task to get a good performance.

3. RELATED WORK

Power-management features are integrated in computer systems of almost every size and class, from handheld devices to large servers [17]. An important feature is the DVFS technique, which trades off performance for power consumption by lowering the operating voltage and frequency [22]. 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 [7, 6, 22] for sequential tasks.

The energy consumption of parallel algorithms for shared memory architectures based on the parallel external memory (PEM) model [1] has been considered in [10]. 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 parallel execution and energy consumption is investigated in [3] by partitioning a parallel algorithm into sequential and parallel regions and computing optimal frequencies for these regions. A task structuring of the parallel algorithms is not considered. Approaches for an energy complexity metric are discussed in [2]. The energy consumption of task-based algorithms with precedence constraints between tasks has been dealt with in [12, 23]. In particular, a scheduling algorithm is presented for adjusting the frequency of the tasks such that the energy consumption is minimized. The task graphs considered are constructed of single-processor tasks, so no task-internal communication is considered.

An energy-oriented evaluation of communication optimization for networks is considered in [8], but the focus lies 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., [4, 13, 21]. These approaches are usually based on heuristics and are not on an analytical model as presented in this work.

4. ANALYTICAL ENERGY MODEL

To capture the energy consumed by a processor, we exploit a well-accepted energy model that has already been used for embedded systems [22], for heterogeneous computing systems [12], or for shared-memory architectures [10]. In this energy model, 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 [7].

The dynamic power consumption can be approximated by

$$P_{dyn} = \alpha \cdot C_L \cdot V^2 \cdot f \qquad (1)$$

where α is the switching probability, C_L is the load capacitance, V is the supply voltage, and f is the operational frequency. In [22] it has been shown that the static power consumption P_{static} can be assumed to be independent of the scaling factor. 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. This section summarizes the energy model with this assumption according to [22].

4.1. 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 for various frequency values. Reducing the frequency by a scaling factor of 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 comsumption:

$$\begin{aligned} \tilde{P}_{dyn} &= \alpha \cdot C_L \cdot V^2 \cdot \tilde{f} \\ &= \alpha \cdot C_L \cdot \beta^2 \cdot \tilde{f}^3 \qquad = \alpha \cdot C_L \cdot \beta^2 \cdot s^{-3} \cdot f^3 \\ &= \alpha \cdot C_L \cdot V^2 \cdot f \cdot s^{-3} \quad = s^{-3} \cdot P_{dyn} \quad . \end{aligned}$$

This shows that the dynamic power is decreased by a factor of s^{-3}. In the following, s is used as a parameter and the dynamic power consumption with scaling factor s is denoted by $P_{dyn}(s)$. Thus, the formula

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

holds. The rest of this section summarizes how to model the energy consumption of a single task executed sequentially on one processor.

For modern processors it only takes a few cycles to change the frequency value [18]. Hence, the switching overhead is small compared to the typical execution time of parallel tasks, which usually requires a large number of cycles for execution, and therefore it can be neglected. This is particularly justified because the frequency value is not changed during the execution of an individual task. For simplicity, the switching overhead can be ignored for the following computations.

4.2. Sequential execution of tasks

The frequency scaling influences the execution time and the energy consumption of task executions in the following way. The sequential execution time $C_T(1)$ of a task $T \in \mathcal{T}$ increases linearly with the scaling factor s, resulting in the execution time $C_T(1) \cdot s$, when the frequency is reduced by a factor of s. With scaling factor $s \geq 1$ as parameter of the energy consumption (i.e., the power consumption multiplied by the execution time, measured in Ws), see [22], the dynamic energy consumption E^T_{dyn} of the task T executed on one processor can be modeled as:

$$E^T_{dyn}(s,1) = P_{dyn}(s) \cdot C_T(1) \cdot s. \tag{3}$$

Using the relation (2) of the dynamic power consumption $P_{dyn}(s)$ for scaling factor $s \geq 1$ to the dynamic power consumption of the unscaled system, i.e. $s = 1$, results in the following relation of the dynamic energy consumption $E^T_{dyn}(s,1)$ for scaling factor $s \geq 1$ to the dynamic energy consumption of the unscaled system:

$$\begin{aligned} E^T_{dyn}(s,1) &= s^{-3} \cdot P_{dyn}(1) \cdot C_T(1) \cdot s \\ &= s^{-2} \cdot E^T_{dyn}(1,1) \quad . \end{aligned} \tag{4}$$

Analogously, the static energy consumption is modeled as:

$$\begin{aligned} E^T_{static}(s,1) &= P_{static} \cdot (C_T(1) \cdot s) \\ &= s \cdot E_{static}(1,1) \end{aligned} \tag{5}$$

with $E_{static}(1,1) = P_{static} \cdot C_T(1)$. According to Equs. (3) and (5), the total energy consumption for the sequential execution of task T on one processor is:

$$\begin{aligned} E^T_{total}(s,1) &= E^T_{dyn}(s,1) + E^T_{static}(s,1) \\ &= (s^{-2} \cdot P_{dyn}(1) + s \cdot P_{static}) \cdot C_T(1) \quad , \end{aligned} \tag{6}$$

expressing the dependence of the energy on the scaling factor.

4.3. Optimal scaling factor

The optimal scaling factor for a sequential execution of tasks can be obtained by considering the power consumption

$$Q_{total}(s) = s^{-2} \cdot P_{dyn}(1) + s \cdot P_{static} \tag{7}$$

of $E^T_{total}(s,1)$ in Equ. (6). Since $C_T(1)$ in Equ. (6) is independent of the scaling factor s [22], the value that minimizes $Q_{total}(s)$ also minimizes $E^T_{total}(s,1)$. The function $Q_{total}(s)$ is convex, because its second derivative $Q''_{total}(s)$ exists and $Q''_{total}(s) \geq 0$. Thus, the optimal scaling factor can be obtained by equating $Q'_{total}(s) = -2 \cdot P_{dyn}(1)/s^{-3} + P_{static}$ to zero. Hence, the optimal scaling factor minimizing the energy consumption $E^T_{total}(s,1)$ is

$$s_{opt} = \left(\frac{2 \cdot P_{dyn}(1)}{P_{static}} \right)^{1/3} . \tag{8}$$

Assuming that $P_{dyn}(1)$ is independent of the computations performed, s_{opt} depends only on the characteristics of the given processor. As example, Fig. 3 shows the resulting power consumption for different realistic values of $z = P_{dyn}(1)/P_{static}$, see, e.g., [5]. The power consumption is normalized with respect to P_{static}, i.e., $Q_{total}(s)/P_{static}$ is depicted. For $z = 2$, for example, $s_{opt} = 1.59$ results.

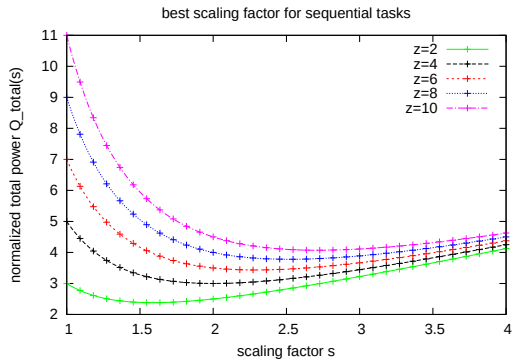

Figure 3. Normalized total power for different relations $z = P_{dyn}(1)/P_{static}$ using different scaling factors s. The optimum scaling factor is given by the Formula (8).

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 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.

5. MODELING ENERGY COMSUMPTION OF PARALLEL TASKS

In this section, we extend the energy model given in the previous section to explore the energy comsumption of parallel tasks with different characteristics.

5.1. Parallel tasks according to Amdahl's law

For the parallel execution of a task, it is assumed that the execution time of each task $T \in \mathcal{T}$ can be subdivided into two parts according to Amdahl's law: one part can be parallelized perfectly and the other part captures computations that cannot be parallelized at all, i.e., the execution time of the second part remains constant, independently of the number of processors used. Using Amdahl's law, the parallel execution time of a task T using q processors is described by

$$C_T(q) = \frac{1-\sigma}{q} \cdot C_T(1) + \sigma \cdot C_T(1) = C_T(1)(\frac{1-\sigma}{q} + \sigma) \quad (9)$$

where σ is the fraction of T that must be executed sequentially; σ is independent of q.

We consider a set of homogeneous processors and assume that all processors executing a specific task T consume the same energy. Thus, the dynamic energy consumption for a parallel execution of task T with q processors and with scaling factor s is described by

$$E^T_{dyn}(s,q) = q \cdot P_{dyn}(s) \cdot C_T(q) \cdot s\,.$$

comm. operation	runtime modeling
single transfer	$T_{s2s}(m) = \tau_1 + t_c \cdot m$
single-broadcast	$T_{sb}(p,m) = \tau_2 \log(p) + t_c \cdot \log(p) \cdot m$
single-accumulation	$T_{acc}(p,m) = \tau_2 \log(p) + t_c \cdot \log(p) \cdot m$
multi-broadcast	$T_{mb}(p,m) = \tau_1 + \tau_2 \cdot p + t_c \cdot p \cdot m$
gather and scatter	$T_g(p,m) = \tau_1 + \tau_2 \cdot p + t_c \cdot p \cdot m$

Table 1. Modeling of communication times for typical communication operations: m is the size of the message exchanged, p is the number of participating processors, τ_1, τ_2, and t_c are machine specific constants.

Using Equ. (9) and then Equ. (3) results in

$$\begin{aligned} E^T_{dyn}(s,q) &= q \cdot P_{dyn}(s) \cdot C_T(1) \cdot ((1-\sigma)/q + \sigma) \cdot s \\ &= q \cdot E^T_{dyn}(s,1) \cdot (1-\sigma)/q + \sigma) \\ &= E^T_{dyn}(s,1) \cdot (1 + (q-1) \cdot \sigma) \quad . \end{aligned}$$

Using Equ. (5), the static energy consumption can be described analogously by

$$\begin{aligned} E^T_{static}(s,q) &= q \cdot P_{static} \cdot (C_T(q) \cdot s) \\ &= E^T_{static}(s,1) \cdot (1 + (q-1) \cdot \sigma) \quad . \end{aligned}$$

Thus, the total energy consumption for the parallel execution of T on q processors depends on the energy consumption on one processor in the following way:

$$E^T_{total}(s,q) = E^T_{total}(s,1)(1 + (q-1) \cdot \sigma) \quad . \qquad (10)$$

The term $(q-1) \cdot \sigma$ can be interpreted as an *energy overhead* for the parallel execution of task T. It increases linearly with the number of processors used to execute T and is larger if the sequential fraction σ of T is larger.

5.2. Parallel tasks with communication

Amdahl's law used in Section 5.1. abstracts from communication costs resulting from communication operations of distributed memory machines. In order to include communication costs, we use the cost formula

$$C_T(q) = C_T(1)((1-\sigma)/q + \sigma) + C^{comm}_T(q) \qquad (11)$$

where $C^{comm}_T(q)$ can be derived from the communication operations performed during the execution of T, i.e., $C^{comm}_T(q)$ is the sum of the execution times of these communication operations. Table 1 describes typical equations for standard communication operations as they are provided by MPI. The parameters t_c, τ_1, and τ_2 that are used in the modeling equations are characteristic for a specific execution platform and can be obtained by corresponding runtime experiments.

As a typical example for a parallel task with communication, we consider a task that computes a matrix-vector product

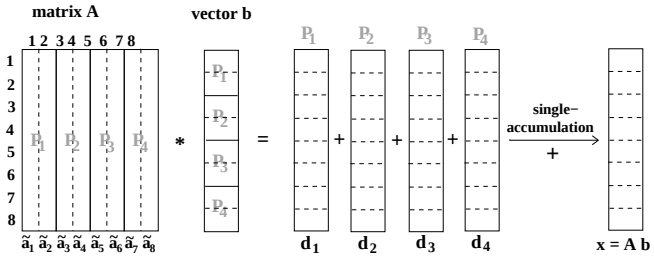

Figure 4. : Illustration of a column-oriented matrix-vector multiplication for an 8×8 matrix, using four processors P_1, P_2, P_3, P_4. Matrix A is distributed column-wise. Each processor P_i computes a vector d_i, $i = 1, \ldots, 4$. The final solution vector x is built by a single-accumulation communication with addition as reduction operation.

$x = A \cdot b$ using a distributed address space. The data distribution used for matrix $A \in \mathbb{R}^{n \times n}$ and vector $b \in \mathbb{R}^n$ determines the communication required. Using a column-wise distribution of matrix A, the matrix-vector multiplication has the form of a linear combination

$$\mathbf{A} \cdot \mathbf{b} = \sum_{j=1}^{n} b_j \tilde{\mathbf{a}}_j$$

of column vectors $\tilde{\mathbf{a}}_1, \ldots, \tilde{\mathbf{a}}_n$ of $\mathbf{A}$ with scalar values $b_1, \ldots, b_n$. In a parallel implementation, each processor P_i, $i \in \{1, \ldots, q\}$ computes that part d_i, $i = 1, \ldots, q$, of the linear combination for which it owns the corresponding columns. For this computation, only a block of elements of vector b is accessed and needs to be stored in the private memory. Since the vectors d_i are stored in different local memories, the final addition to the complete matrix-vector product requires an accumulation operation with addition as reduction, see Fig. 4 for an illustration.

The sequential execution time of the matrix-vector multiplication can be approximated by $C_T(1) = \gamma \cdot n^2$ with an appropriate constant γ. Taking single-accumulation communication into account, the parallel execution time using q processors can be modeled by

$$C_T(q) = \gamma \cdot n^2/q + \log(q)(\tau_2 + t_c \cdot n). \tag{12}$$

This equation uses the modeling equation of Table 1 for the accumulation for which each processor contributes n values. With the execution time from Equ. (12) and using Equ. (3), the dynamic energy consumption for q processors can be modeled by

$$\begin{aligned} &E^T_{dyn}(s,q) = q \cdot P_{dyn}(s) \cdot C_T(q) \cdot s \\ &= q \cdot P_{dyn}(s) \cdot (C_T(1)/q + \log(q)(\tau_2 + t_c \cdot n)) \cdot s \\ &= E^T_{dyn}(s,1) + q \cdot s^{-2} \cdot P_{dyn}(1) \cdot \log(q) \cdot (\tau_2 + t_c \cdot n) \end{aligned}$$

The second term can be considered as *dynamic power consumption overhead* due to communication. The static power consumption for q processors is modeled analogously:

$$\begin{aligned} &E^T_{static}(s,q) = q \cdot P_{static} \cdot C_T(q) \cdot s \\ &= E^T_{static}(s,1) + q \cdot s \cdot P_{static} \cdot \log(q) \cdot (\tau_2 + t_c \cdot n) \end{aligned}$$

The second term represents the *static power consumption overhead* due to communication. The total energy consumption is then modeled as:

$$\begin{aligned} &E^T_{total}(s,q) = E^T_{dyn}(s,q) + E^T_{static}(s,q) \\ &= E^T_{dyn}(s,1) + E^T_{static}(s,1) + \\ &\quad (s^{-2} \cdot P_{dyn}(1) + s \cdot P_{static}) \cdot q \cdot \log(q) \cdot (\tau_2 + t_c \cdot n), \end{aligned} \tag{13}$$

which shows that the energy consumption overhead increases with $q \log q$. Considering Equs. (12) and (13), it can be observed that this overhead is caused by the task-internal communication, which is one single-accumulation for the example considered.

In the general case, several communication operations may be executed during the execution of a task T. The execution times of these communication operations add up to $C^{comm}_T(q)$. Correspondingly, the total energy consumption of q processors scaled with scaling factor s is

$$\begin{aligned} E^T_{total}(s,q) &= q \cdot (P_{dyn}(s) + P_{static}) \cdot \\ &\quad \left(C_T(1) \cdot \left(\tfrac{1-\sigma}{q} + \sigma \right) + C^{comm}_T(q) \right) \cdot s \\ &= (s^{-2} P_{dyn}(s) + P_{static}) \cdot \\ &\quad (C_T(1) \cdot (1 + (q-1) \cdot \sigma) + q \cdot C^{comm}_T(q)) \\ &= Q_{total}(s) \cdot \\ &\quad (C_T(1) \cdot (1 + (q-1) \cdot \sigma) + q \cdot C^{comm}_T(q)) \end{aligned}$$

For a fixed number of processors, $E^T_{total}(s,q)$ is minimized at the minimum of the function $Q_{total}(s)$, i.e., for the scaling factor s_{opt} from Equ. 8. Thus, assuming the parallel execution time to be modeled by Equ. 11, sequential tasks and parallel tasks have the same optimal scaling factor.

6. ENERGY CONSUMPTION FOR TASK-BASED EXECUTIONS

Based on the energy model for a single parallel task derived in the previous section, we now consider the energy consumption in the task parallel program model from Sect. 2.

6.1. Energy consumption of composed tasks

In the programming model of parallel tasks, two tasks can be composed in a consecutive or concurrent execution. For a consecutive execution of two tasks $T_1 \circ T_2$, all processors available are used for T_1 first and then for T_2. Thus, the energy consumptions of T_1 and T_2 are added to get the total energy consumption of $T_1 \circ T_2$:

$$E^{T_1 \circ T_2}_{total}(s,q) = E^{T_1}_{total}(s,q) + E^{T_2}_{total}(s,q) \tag{14}$$

Again, s_{opt} from Equ. (8) leads to a minimum energy consumption if q is considered to be fixed. For tasks that are independent of each other (i.e., $T_1||T_2$), T_1 and T_2 can be executed consecutively, again leading to the energy consumption in Equ. (14). However, T_1 and T_2 can also be executed concurrently by two disjoint groups G_1 and G_2 of processors with $|G_1| = q_1$, $|G_2| = q_2$, $1 \leq q_1, q_2 \leq q$, and $q_1 + q_2 = q$. The resulting energy consumption is

$$E_{total}^{T_1\|T_2}(s,q_1,q_2) = E_{total}^{T_1}(s,q_1) + E_{total}^{T_2}(s,q_2),$$

assuming that q_1 and q_2 can be chosen such that T_1 and T_2 are finished at about the same time. In the next subsection, we show that a concurrent execution of independent tasks leads to a smaller energy consumption than a consecutive execution.

6.2. Energy consumption of concurrent tasks

As example, we consider the concurrent execution of two tasks where each of the tasks computes a matrix-vector product. Both matrices are of of the same size. For each of these tasks, the energy consumption is given by Equ. (13). Using these energy values for both tasks T_1 and T_2 results in the following energy consumption of $T_1 \parallel T_2$:

$$E_{total}^{T_1\|T_2}(s,q_1,q_2) = E_1(s) + Q_{total}(s) \cdot (q_1 \cdot \log q_1 + q_2 \cdot \log q_2)(\tau_2 + t_c \cdot n)$$

with

$$E_1(s) := E_{total}^{T_1\|T_2}(s,1,1) \quad = E_{dyn}^{T_1}(s,1) + E_{static}^{T_1}(s,1) + E_{dyn}^{T_2}(s,1) + E_{static}^{T_2}(s,1)$$

and $Q_{total}(s)$ from Equ. (7). In $E_{total}^{T_1\|T_2}(s,q_1,q_2)$, only the expression $f(q_1,q_2) := q_1 \cdot \log q_1 + q_2 \cdot \log q_2$ depends on both q_1 and q_2 with $q_1 + q_2 = q$. Using $q_2 = q - q_1$ with q constant leads to the function $\tilde{f}(q_1) := q_1 \cdot \log q_1 + (q - q_1) \cdot \log(q - q_1)$. Building the derivative and setting $\tilde{f}'(q_1) = 0$ yields that $\tilde{f}(q_1)$ is minimized for $q_1 = q/2$. This means that the same number of processors, i.e., $q_1 = q_2$, should be used for T_1 and T_2. Since $q_1 \cdot \log q_1 + q_2 \cdot \log q_2 = 2 \cdot q_1 \cdot \log q_1 = q \cdot \log(q/2)$ for $q_1 = q_2 = q/2$, the resulting energy consumption is

$$E_{total}^{T_1\|T_2}(s,q_1,q_1) = E_1(s) + Q_{total}(s)(q \cdot \log(q/2))(\tau_2 + t_c \cdot n) \tag{15}$$

On the other hand, from Equ. (14) it can be seen that the energy consumption for a consecutive execution of T_1 and T_2 with q processors each results in

$$E_{total}^{T_1\circ T_2}(s,q) = E_1(s) + Q_{total}(s)(2 \cdot q \cdot \log q)(\tau_2 + t_c \cdot n), \tag{16}$$

i.e., for the same scaling factor used, the energy consumption overhead due to communication is about twice as large as for the concurrent execution of T_1 and T_2 in Equ. (15). Thus, from the energy consumption perspective, it is advantageous to perform a concurrent execution. For both cases, a minimum energy consumption is obtained for scaling factor s_{opt} from Equ (8).

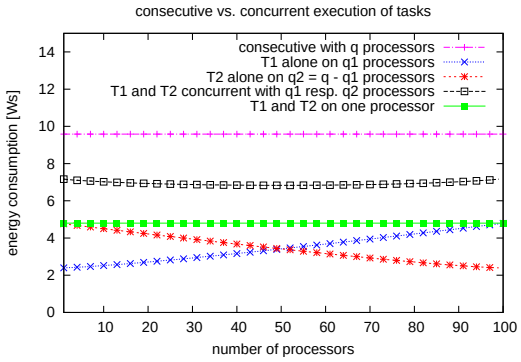

Figure 5. Simulated energy consumption for different execution orders of two parallel tasks, each performing a matrix-vector multiplication.

Figure 5 shows the simulated energy consumption for a concurrent execution of two tasks each performing a matrix-vector multiplication using different numbers of processors and compares it to a consecutive execution. For the execution, $q = 100$ processors with power consumption $P_{static} = 4W$ and $P_{dyn}(1) = 20W$ are used; these are realistic values for current processor chips [5]. For the communication, the following values have been used: $\tau_2 = 50 \cdot 10^{-6} sec$, $t_c = 20 \cdot 10^{-9} sec$, assuming an effective bandwidth of about $200MB/sec$. The matrix size is 5000×5000 and for each computation, a total execution time of $t_{op} = 4 \cdot 10^{-9} sec$ is assumed, including memory accesses.

The diagram shows that the purely sequential execution produces the smallest amount of energy of about 4.8 Ws. For a parallel execution, more energy is needed, at least about 6.8 Ws for using 50 processors for each of the two tasks. A consecutive execution with 100 processors for each task uses significantly more energy of about 9.6 Ws. For other realistic values, the results obtained are similar.

In the general case, T_1 and T_2 are different tasks performing different computations and requiring different task-internal communication operations. The energy consumption is then

$$E_{total}^{T_1\|T_2}(s,q_1,q_2) = E_1(s) + Q_{total}(s)(q_1 \cdot C_{T_1}^{comm}(q_1) + q_2 \cdot C_{T_2}^{comm}(q_2)), \tag{17}$$

Thus, from the energy consumption perspective, q_1 and q_2 with $q_1 + q_2 = q$ should be selected such that $f(q_1,q_2) = q_1 \cdot C_{T_1}^{comm}(q_1) + q_2 \cdot C_{T_2}^{comm}(q_2)$ is minimized, i.e., the subdivision of the q processors into two groups solely depends on the internal communication of T_1 and T_2, and not on the com-

putations performed. In contrast, if the execution times for $T_1||T_2$ (and not the energy) are considered, the computations performed by T_1 and T_2 must definitely be taken into consideration when determining q_1 and q_2.

The energy modeling result obtained for two concurrent tasks can be generalized to an arbitrary number of independent tasks $T_1, \ldots, T_k$ that can be executed concurrently. The sizes of the k disjoint processor groups assigned should be chosen such that all tasks are finished at about the same time. Then again, s_{opt} from Equ (8 can be used to obtain a minimum energy consumption.

7. CONCLUSIONS

This article has considered the execution of task-based programs from an energy-consumption perspective using a well-accepted analytical energy model. In particular, a task-based programming model has been exploited in which each task can be executed by multiple processors in parallel. Using a distributed address space and taking task-internal communication into consideration, it has been shown that the resulting energy overhead is smaller, if independent tasks are executed concurrently by disjoint sets of processors compared to a consecutive execution. Moreover, the energy model can be used to compute a scaling factor for each task of a set of concurrently executed tasks, such that the overall energy consumption is minimized by avoiding waiting times.

REFERENCES

[1] L. Arge, M.T. Goodrich, M. Nelson, and N. Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *SPAA '08: Proc. of the 20th Ann. Symp. on Parallelism in Algorithms and Architectures*, pages 197–206. ACM, 2008.

[2] B.D. Bingham and M.R. Greenstreet. 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*, pages 143–152. IEEE Computer Society, 2008.

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

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

[5] Intel. *Quad-Core Intel Xeon Processor 5400 Series Datasheet*, 2008.

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

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

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

[9] D.J. Kerbyson and A. Hoisie. A practical approach to performance analysis and modeling of large-scale systems. In *Proc. ACM/IEEE SC2006 Conference*, page 206, 2006.

[10] V.A. Korthikanti and G. Agha. Towards optimizing energy costs of algorithms for shared memory architectures. In *SPAA '10: Proc. of the 22nd ACM Symp. on Parallelism in Algorithms and Architectures*, pages 157–165. ACM, 2010.

[11] M. Kühnemann, T. Rauber, and G. Rünger. Performance Modelling for Task-Parallel Programs. In *Proc. of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, pages 148–154, 2002.

[12] Y.C. Lee and A.Y. Zomaya. 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*, pages 92–99. IEEE Computer Society, 2009.

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

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

[15] T. Rauber and G. Rünger. A Transformation Approach to Derive Efficient Parallel Implementations. *IEEE Transactions on Software Engineering*, 26(4):315–339, 2000.

[16] T. Rauber and G. Rünger. Tlib - A Library to Support Programming with Hierarchical Multi-Processor Tasks. *Journal of Parallel and Distributed Computing*, 65(3):347–360, 2005.

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

[18] A. Sinkar and N. Kim. Analyzing Potential Total Power Reduction with Adaptive Voltage Positioning Optimized for Multicore Processors. In *Proc. IEEE/ACM Int. Symp. on Low Power Electronic Design (ISLPED)*, 2009.

[19] N. Vydyanathan, S. Krishnamoorthy, G. Sabin, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz. Locality Conscious Processor Allocation and Scheduling for Mixed Parallel Applications. In *Proc. of the IEEE Int. Conf. on Cluster Computing*. IEEE, 2006.

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

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

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

[23] Z. Zong, X. Qin, X. Ruan, K. Bellam, M. Nijim, and M. Alghamdi. Energy-Efficient Scheduling for Parallel Applications Running on Heterogeneous Clusters. In *ICPP '07: Proc. of the 2007 Int. Conf. on Parallel Processing*, page 19. IEEE Computer Society, 2007.