

Energy-Aware Algorithms for Task Graph Scheduling, Replica Placement and Checkpoint Strategies

Guillaume Aupy, Anne Benoit, Paul Renaud-Goud and Yves Robert

1 Introduction

The *energy consumption* of computational platforms has recently become a critical problem, both for economic and environmental reasons [35]. To reduce energy consumption, processors can run at different speeds. Faster speeds allow for a faster execution, but they also lead to a much higher (superlinear) power consumption. Energy-aware scheduling aims at minimizing the energy consumed during the execution of the target application, both for computations and for communications. The price to pay for a lower energy consumption usually is a much larger execution time, so the energy-aware approach makes better sense when coupled with some prescribed performance bound. In other words, we have a bi-criteria optimization problem, with one objective being energy minimization, and the other being performance-related.

In this chapter, we discuss several problems related to data centers, for which energy consumption is a crucial matter. Indeed, statistics showed that in 2012, some data centers consume more electricity than 250,000 european houses. If the *cloud* was a country, it would be ranked as the fifth world-wide rank in terms of demands in electricity, and the need is expected to be multiplied by three before 2020. We

G. Aupy (✉) · A. Benoit · Y. Robert
LIP, Ecole Normale Supérieure de Lyon, Lyon, France
e-mail: Guillaume.Aupy@ens-lyon.fr

A. Benoit · Y. Robert
Institut Universitaire de France, Paris, France
e-mail: Anne.Benoit@ens-lyon.fr

P. Renaud-Goud
Chalmers University of technology, Gothenburg, Sweden
e-mail: goud@chalmers.se

Y. Robert
University Tennessee Knoxville, Knoxville, USA
e-mail: Yves.Robert@ens-lyon.fr

focus mainly on the energy consumption of processors, although a lot of electricity is now devoted to cooling the machines, and also for network communications.

Energy models are introduced in Sect. 2. Depending on the different research areas, several different energy models are considered, but they all share the same core assumption: there is a static energy consumption, which is independent on the speed at which a processor is running, and a dynamic energy consumption, which increases superlinearly with the speed. The most common models for speeds are either to use continuous speeds in a given interval, or to consider a set of discrete speeds (the latter being more realistic for actual processors). We discuss further variants of the discrete model: in the VDD-hopping model, the speed of a task can be changed during execution, hence allowing to simulate the continuous case; the incremental model is similar to the discrete model with the additional assumption that the different speeds are spaced regularly. Finally, we propose a literature survey on energy models, and we provide an example to compare models.

The first case study is about task graph scheduling (see Sect. 3). We consider a task graph to be executed on a set of processors. We assume that the mapping is given, say by an ordered list of tasks to execute on each processor, and we aim at optimizing the energy consumption while enforcing a prescribed bound on the execution time. While it is not possible to change the allocation of a task, it is possible to change its speed. Rather than using a local approach such as backfilling, we consider the problem as a whole and study the impact of several speed variation models on its complexity. For continuous speeds, we give a closed-form formula for trees and series-parallel graphs, and we cast the problem into a geometric programming problem for general directed acyclic graphs. We show that the classical dynamic voltage and frequency scaling (DVFS) model with discrete speeds leads to an NP-complete problem, even if the speeds are regularly distributed (an important particular case in practice, which we analyze as the incremental model). On the contrary, the VDD-hopping model leads to a polynomial solution. Finally, we provide an approximation algorithm for the incremental model, which we extend for the general DVFS model.

Then in Sect. 4, we discuss a variant of the replica placement problem aiming at an efficient power management. We study optimal strategies to place replicas in tree networks, with the double objective to minimize the total cost of the servers, and/or to optimize power consumption. The client requests are known beforehand, and some servers are assumed to pre-exist in the tree. Without power consumption constraints, the total cost is an arbitrary function of the number of existing servers that are reused, and of the number of new servers. Whenever creating and operating a new server has higher cost than reusing an existing one (which is a very natural assumption), cost optimal strategies have to trade-off between reusing resources and load-balancing requests on new servers. We provide an optimal dynamic programming algorithm that returns the optimal cost, thereby extending known results from Wu, Lin and Liu [33, 43] without pre-existing servers. With power consumption constraints, we assume that servers operate under a set of M different speeds depending upon the number of requests that they have to process. In practice, M is a small number, typically 2 or 3, depending upon the number of allowed voltages [24, 23]. Power consumption includes a static part, proportional to the total number of servers, and

a dynamic part, proportional to a constant exponent of the server speed, which depends upon the model for power. The cost function becomes a more complicated function that takes into account reuse and creation as before, but also upgrading or downgrading an existing server from one speed to another. We show that with an arbitrary number of speeds, the power minimization problem is NP-complete, even without cost constraint, and without static power. Still, we provide an optimal dynamic programming algorithm that returns the minimal power, given a threshold value on the total cost; it has exponential complexity in the number of speeds M , and its practical usefulness is limited to small values of M . However, experiments conducted with this algorithm show that it can process large trees in reasonable time, despite its worst-case complexity.

The last case study investigates checkpointing strategies (see Sect. 5). Nowadays, high performance computing is facing a major challenge with the increasing frequency of failures [18]. There is a need to use fault tolerance or resilience mechanisms to ensure the efficient progress and correct termination of the applications in the presence of failures. A well-established method to deal with failures is *checkpointing*: a checkpoint is taken at the end of the execution of each chunk of work. During the checkpoint, we check for the accuracy of the result; if the result is not correct, due to a transient failure (such as a memory error or software error), the chunk is re-executed. This model with transient failures is one of the most used in the literature, see for instance [17, 48]. In this section, we aim at minimizing the energy consumption when executing a divisible workload under a bound on the total execution time, while resilience is provided through checkpointing. We discuss several variants of this multi-criteria problem. Given the workload, we need to decide how many chunks to use, what are the sizes of these chunks, and at which speed each chunk is executed (under the continuous model). Furthermore, since a failure may occur during the execution of a chunk, we also need to decide at which speed a chunk should be re-executed in the event of a failure. The goal is to minimize the expectation of the total energy consumption, while enforcing a deadline on the execution time, that should be met either in expectation (soft deadline), or in the worst case (hard deadline). For each problem instance, we propose either an exact solution, or a function that can be optimized numerically.

Finally, we provide concluding remarks in Sect. 6.

2 Energy Models

As already mentioned, to help reduce energy dissipation, processors can run at different speeds. Their power consumption is the sum of a static part (the cost for a processor to be turned on, and the leakage power) and a dynamic part, which is a strictly convex function of the processor speed, so that the execution of a given amount of work costs more power if a processor runs in a higher speed [23]. More precisely, a processor running at speed s dissipates s^3 watts [4, 12, 15, 25, 38] per time-unit, hence consumes $s^3 \times d$ joules when operated during d units of time. Faster

speeds allow for a faster execution, but they also lead to a much higher (superlinear) power consumption.

In this section, we survey different models for dynamic energy consumption, taken from the literature. These models are categorized as follows:

CONTINUOUS model. Processors can have arbitrary speeds, and can vary them continuously within the interval $[s_{min}, s_{max}]$. This model is unrealistic (any possible value of the speed, say $\sqrt{e^\pi}$, cannot be obtained) but it is theoretically appealing [5]. In the CONTINUOUS model, a processor can change its speed at any time during execution.

DISCRETE model. Processors have a discrete number of predefined speeds, which correspond to different voltages and frequencies that the processor can be subjected to [36]. These speeds are denoted as s_1, \dots, s_m . Switching speeds is not allowed during the execution of a given task, but two different tasks scheduled on a same processor can be executed at different speeds.

VDD-HOPPING model. This model is similar to the DISCRETE one, with a set of different speeds s_1, \dots, s_m , except that switching speeds during the execution of a given task is allowed: any rational speed can be simulated, by simply switching, at the appropriate time during the execution of a task, between two consecutive speeds [34]. In the VDD-HOPPING model, the energy consumed during the execution of one task is the sum, on each time interval with constant speed s , of the energy consumed during this interval at speed s .

INCREMENTAL model. In this variant of the DISCRETE model, there is a value δ that corresponds to the minimum permissible speed increment, induced by the minimum voltage increment that can be achieved when controlling the processor CPU. Hence, possible speed values are obtained as $s = s_{min} + i \times \delta$, where i is an integer such that $0 \leq i \leq \frac{s_{max} - s_{min}}{\delta}$. Admissible speeds lie in the interval $[s_{min}, s_{max}]$. This model aims at capturing a realistic version of the DISCRETE model, where the different speeds are spread regularly between $s_1 = s_{min}$ and $s_m = s_{max}$, instead of being arbitrarily chosen. It is intended as the modern counterpart of a potentiometer knob.

After the literature survey in Sect. 2.1, we provide a simple example in Sect. 2.2, in order to illustrate the different models.

2.1 Literature Survey

Reducing the energy consumption of computational platforms is an important research topic, and many techniques at the process, circuit design, and micro-architectural levels have been proposed [22, 30, 32]. The dynamic voltage and frequency scaling (DVFS) technique has been extensively studied, since it may lead to efficient energy/performance trade-offs [5, 14, 20, 26, 29, 42, 45]. Current microprocessors (for instance, from AMD [1] and Intel [24]) allow the speed to be set dynamically. Indeed, by lowering supply voltage, hence processor clock frequency,

it is possible to achieve important reductions in power consumption, without necessarily increasing the execution time. We first discuss different optimization problems that arise in this context, then we review energy models.

2.1.1 DVFS and Optimization Problems

When dealing with energy consumption, the most usual optimization function consists of minimizing the energy consumption, while ensuring a deadline on the execution time (i.e., a real-time constraint), as discussed in the following papers.

In [36], Okuma et al. demonstrate that voltage scaling is far more effective than the shutdown approach, which simply stops the power supply when the system is inactive. Their target processor employs just a few discretely variable voltages. De Langen and Juurlink [31] discuss leakage-aware scheduling heuristics that investigate both DVS and processor shutdown, since static power consumption due to leakage current is expected to increase significantly. Chen et al. [13] consider parallel sparse applications, and they show that when scheduling applications modeled by a directed acyclic graph with a well-identified critical path, it is possible to lower the voltage during non-critical execution of tasks, with no impact on the execution time. Similarly, Wang et al. [42] study the slack time for non-critical jobs, they extend their execution time and thus reduce the energy consumption without increasing the total execution time. Kim et al. [29] provide power-aware scheduling algorithms for bag-of-tasks applications with deadline constraints, based on dynamic voltage scaling. Their goal is to minimize power consumption as well as to meet the deadlines specified by application users.

For real-time embedded systems, slack reclamation techniques are used. Lee and Sakurai [32] show how to exploit slack time arising from workload variation, thanks to a software feedback control of supply voltage. Prathipati [37] discusses techniques to take advantage of run-time variations in the execution time of tasks; the goal is to determine the minimum voltage under which each task can be executed, while guaranteeing the deadlines of each task. Then, experiments are conducted on the Intel StrongArm SA-1100 processor, which has eleven different frequencies, and the Intel PXA250 XScale embedded processor with four frequencies. In [44], the goal of Xu et al. is to schedule a set of independent tasks, given a worst case execution cycle (WCEC) for each task, and a global deadline, while accounting for time and energy penalties when the processor frequency is changing. The frequency of the processor can be lowered when some slack is obtained dynamically, typically when a task runs faster than its WCEC. Yang and Lin [45] discuss algorithms with preemption, using DVS techniques; substantial energy can be saved using these algorithms, which succeed to claim the static and dynamic slack time, with little overhead.

Since an increasing number of systems are powered by batteries, maximizing battery life also is an important optimization problem. Battery-efficient systems can be obtained with similar techniques of dynamic voltage and frequency scaling,

as described by Lahiri et al. in [30]. Another optimization criterion is the energy-delay product, since it accounts for a trade-off between performance and energy consumption, as for instance discussed by Gonzalez and Horowitz in [21].

2.1.2 Energy Models

Several energy models are considered in the literature, and they can all be categorized in one of the four models investigated in this paper, i.e., CONTINUOUS, DISCRETE, VDD-HOPPING or INCREMENTAL.

The CONTINUOUS model is used mainly for theoretical studies. For instance, Yao et al. [46], followed by Bansal et al. [5], aim at scheduling a collection of tasks (with release time, deadline and amount of work), and the solution is the time at which each task is scheduled, but also, the speed at which the task is executed. In these papers, the speed can take any value, hence following the CONTINUOUS model.

We believe that the most widely used model is the DISCRETE one. Indeed, processors have currently only a few discrete number of possible frequencies [1, 24, 36, 37]. Therefore, most of the papers discussed above follow this model. Some studies exploit the continuous model to determine the smallest frequency required to run a task, and then choose the closest upper discrete value, as for instance [37] and [47].

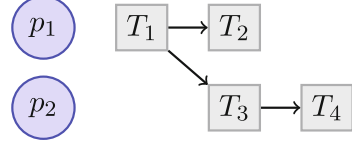
Recently, a new local dynamic voltage scaling architecture has been developed, based on the VDD-HOPPING model [6, 7, 34]. It was shown in [32] that significant power can be saved by using two distinct voltages, and architectures using this principle have been developed (see for instance [28]). Compared to traditional power converters, a new design with no needs for large passives or costly technological options has been validated in a STMicroelectronics CMOS 65-nm low-power technology [34].

The INCREMENTAL model was introduced in [2]. The main rationale is that future technologies may well have an increased number of possible frequencies, and these will follow a regular pattern. For instance, note that the SA-1100 processor, considered in [37], has eleven frequencies that are equidistant, i.e., they follow the INCREMENTAL model. Lee and Sakurai [32] exploit discrete levels of clock frequency as f , $f/2$, $f/3$, ..., where f is the master (i.e., the higher) system clock frequency. This model is closer to the DISCRETE model, although it exhibits a regular pattern similarly to the INCREMENTAL model.

2.2 Example

Energy-aware scheduling aims at minimizing the energy consumed during the execution of the target application. Obviously, it makes better sense only if it is coupled with some performance bound to achieve. For instance, whenever static energy can

Fig. 1 Execution graph for the example



be neglected, the optimal solution always is to run each processor at the slowest possible speed. In the following, we do neglect static energy and discuss how to minimize dynamic energy consumption when executing a small task graph onto processors.

Consider an application with four tasks of costs $w_1 = 3$, $w_2 = 2$, $w_3 = 1$ and $w_4 = 2$, and three precedence constraints, as shown in Fig. 1. We assume that T_1 and T_2 are allocated, in this order, onto processor P_1 , while T_3 and T_4 are allocated, in this order, on processor P_2 . The deadline on the execution time is $D = 1.5$.

We set the minimum and maximum speeds to $s_{min} = 0$ and $s_{max} = 6$ for the CONTINUOUS model. For the DISCRETE and VDD-HOPPING models, we use the set of speeds $s_1^{(d)} = 2$, $s_2^{(d)} = 5$ and $s_3^{(d)} = 6$. Finally, for the INCREMENTAL model, we set $\delta = 2$, $s_{min} = 2$ and $s_{max} = 6$, so that possible speeds are $s_1^{(i)} = 2$, $s_2^{(i)} = 4$ and $s_3^{(i)} = 6$. We aim at finding the optimal execution speed s_i for each task T_i ($1 \leq i \leq 4$), i.e., the values of s_i that minimize the energy consumption.

With the CONTINUOUS model, the optimal speeds are non rational values, and we obtain:

$$s_1 = \frac{2}{3}(3 + 35^{1/3}) \simeq 4.18; s_2 = s_1 \times \frac{2}{35^{1/3}} \simeq 2.56; s_3 = s_4 = s_1 \times \frac{3}{35^{1/3}} \simeq 3.83.$$

Note that all speeds are in the interval $[s_{min}, s_{max}]$. These values are obtained thanks to the formulas derived in Sect. 3.2 below. The energy consumption is then $E_{opt}^{(c)} = \sum_{i=1}^4 w_i \times s_i^2 = 3.s_1^2 + 2.s_2^2 + 3.s_3^2 \simeq 109.6$. The execution time is $\frac{w_1}{s_1} + \max\left(\frac{w_2}{s_2}, \frac{w_3+w_4}{s_3}\right)$, and with this solution, it is equal to the deadline D (actually, both processors reach the deadline, otherwise we could slow down the execution of one task).

For the DISCRETE model, if we execute all tasks at speed $s_2^{(d)} = 5$, we obtain an energy $E = 8 \times 5^2 = 200$. A better solution is obtained with $s_1 = s_3^{(d)} = 6$, $s_2 = s_3 = s_1^{(d)} = 2$ and $s_4 = s_2^{(d)} = 5$, which turns out to be optimal: $E_{opt}^{(d)} = 3 \times 36 + (2 + 1) \times 4 + 2 \times 25 = 170$. Note that $E_{opt}^{(d)} > E_{opt}^{(c)}$, i.e., the optimal energy consumption with the DISCRETE model is much higher than the one achieved with the CONTINUOUS model. Indeed, in this case, even though the first processor executes during $3/6 + 2/2 = D$ time units, the second processor remains idle since $3/6 + 1/2 + 2/5 = 1.4 < D$. The problem turns out to be NP-hard (see Sect. 3.3.2), and the solution was found by performing an exhaustive search.

With the VDD-HOPPING model, we set $s_1 = s_2^{(d)} = 5$; for the other tasks, we run part of the time at speed $s_2^{(d)} = 5$, and part of the time at speed $s_1^{(d)} = 2$ in order to use the idle time and lower the energy consumption. T_2 is executed at speed $s_1^{(d)}$ during time $\frac{5}{6}$ and at speed $s_2^{(d)}$ during time $\frac{2}{30}$ (i.e., the first processor executes during time

$3/5 + 5/6 + 2/30 = 1.5 = D$, and all the work for T_2 is done: $2 \times 5/6 + 5 \times 2/30 = 2 = w_2$). T_3 is executed at speed $s_2^{(d)}$ (during time $1/5$), and finally T_4 is executed at speed $s_1^{(d)}$ during time 0.5 and at speed $s_2^{(d)}$ during time $1/5$ (i.e., the second processor executes during time $3/5 + 1/5 + 0.5 + 1/5 = 1.5 = D$, and all the work for T_4 is done: $2 \times 0.5 + 5 \times 1/5 = 2 = w_4$). This set of speeds turns out to be optimal (i.e., it is the optimal solution of the linear program introduced in Sect. 3.3.1), with an energy consumption $E_{opt}^{(v)} = (3/5 + 2/30 + 1/5 + 1/5) \times 5^3 + (5/6 + 0.5) \times 2^3 = 144$. As expected, $E_{opt}^{(c)} \leq E_{opt}^{(v)} \leq E_{opt}^{(d)}$, i.e., the VDD-HOPPING solution stands between the optimal CONTINUOUS solution, and the more constrained DISCRETE solution.

For the INCREMENTAL model, the reasoning is similar to the DISCRETE case, and the optimal solution is obtained by an exhaustive search: all tasks should be executed at speed $s_2^{(i)} = 4$, with an energy consumption $E_{opt}^{(i)} = 8 \times 4^2 = 128 > E_{opt}^{(c)}$. It turns out to be better than DISCRETE and VDD-HOPPING, since it has different discrete values of energy that are more appropriate for this example.

3 Minimizing the Energy of a Schedule

In this section, we investigate energy-aware scheduling strategies for executing a task graph on a set of processors. The main originality is that we assume that the mapping of the task graph is given, say by an ordered list of tasks to execute on each processor. There are many situations in which this problem is important, such as optimizing for legacy applications, or accounting for affinities between tasks and resources, or even when tasks are pre-allocated [39], for example for security reasons. In such situations, assume that a list-schedule has been computed for the task graph, and that its execution time should not exceed a deadline D . We do not have the freedom to change the assignment of a given task, but we can change its speed to reduce energy consumption, provided that the deadline D is not exceeded after the speed change. Rather than using a local approach such as backfilling [37, 42], which only reclaims gaps in the schedule, we consider the problem as a whole, and we assess the impact of several speed variation models on its complexity. We give the main complexity results without proofs (refer to [2] for details).

3.1 Optimization Problem

Consider an application task graph $\mathcal{G} = (V, \mathcal{E})$, with $n = |V|$ tasks denoted as $V = \{T_1, T_2, \dots, T_n\}$, and where the set \mathcal{E} denotes the precedence edges between tasks. Task T_i has a cost w_i for $1 \leq i \leq n$. We assume that the tasks in \mathcal{G} have been allocated onto a parallel platform made up of identical processors. We define the *execution graph* generated by this allocation as the graph $G = (V, E)$, with the following augmented set of edges:

- $\mathcal{E} \subseteq E$: if an edge exists in the precedence graph, it also exists in the execution graph;
- if T_1 and T_2 are executed successively, in this order, on the same processor, then $(T_1, T_2) \in E$.

The goal is to minimize the energy consumed during the execution while enforcing a deadline D on the execution time. We formalize the optimization problem in the simpler case where each task is executed at constant speed. This strategy is optimal for the CONTINUOUS model (by a convexity argument) and for the DISCRETE and INCREMENTAL models (by definition). For the VDD-HOPPING model, we reformulate the problem in Sect. 3.3.1. Let d_i be the duration of the execution of task T_i , t_i its completion time, and s_i the speed at which it is executed. We obtain the following formulation of the MINENERGY(G, D) problem, given an execution graph $G = (V, E)$ and a deadline D ; the s_i values are variables, whose values are constrained by the energy model:

$$\begin{aligned}
 & \text{Minimize} && \sum_{i=1}^n s_i^3 \times d_i \\
 & \text{subject to} && \begin{aligned}
 & \text{(i)} && w_i = s_i \times d_i \text{ for each task } T_i \in V \\
 & \text{(ii)} && t_i + d_j \leq t_j \text{ for each edge } (T_i, T_j) \in E \\
 & \text{(iii)} && t_i \leq D \text{ for each task } T_i \in V
 \end{aligned}
 \end{aligned} \tag{1}$$

Constraint (i) states that the whole task can be executed in time d_i using speed s_i . Constraint (ii) accounts for all dependencies, and constraint (iii) ensures that the execution time does not exceed the deadline D . The energy consumed throughout the execution is the objective function. It is the sum, for each task, of the energy consumed by this task, as we detail in the next section. Note that $d_i = w_i/s_i$, and therefore the objective function can also be expressed as $\sum_{i=1}^n s_i^2 \times w_i$.

3.2 The CONTINUOUS Model

With the CONTINUOUS model, processor speeds can take any value between s_{min} and s_{max} . We assume for simplicity that $s_{min} = 0$, i.e., there is no minimum speed. First we prove that, with this model, the processors do not change their speed during the execution of a task:

Lemma 1 (constant speed per task) *With the CONTINUOUS model, each task is executed at constant speed, i.e., a processor does not change its speed during the execution of a task.*

We derive in Sect. 3.2.1 the optimal speed values for special execution graph structures, expressed as closed form algebraic formulas, and we show that these values may be irrational (as already illustrated in the example in Sect. 2.2). Finally, we formulate the problem for general DAGs as a convex optimization program in Sect. 3.2.2.

3.2.1 Special Execution Graphs

Consider the problem of minimizing the energy of n independent tasks (i.e., each task is mapped onto a distinct processor, and there are no precedence constraints in the execution graph), while enforcing a deadline D .

Proposition 1 (independent tasks) *When G is composed of independent tasks $\{T_1, \dots, T_n\}$, the optimal solution to $\text{MINENERGY}(G, D)$ is obtained when each task T_i ($1 \leq i \leq n$) is computed at speed $s_i = \frac{w_i}{D}$. If there is a task T_i such that $s_i > s_{\max}$, then the problem has no solution.*

Consider now the problem with a linear chain of tasks. This case corresponds for instance to n independent tasks $\{T_1, \dots, T_n\}$ executed onto a single processor. The execution graph is then a linear chain (order of execution of the tasks), with $T_i \rightarrow T_{i+1}$, for $1 \leq i < n$.

Proposition 2 (linear chain) *When G is a linear chain of tasks, the optimal solution to $\text{MINENERGY}(G, D)$ is obtained when each task is executed at speed $s = \frac{W}{D}$, with $W = \sum_{i=1}^n w_i$. If $s > s_{\max}$, then there is no solution.*

Corollary 1 *A linear chain with n tasks is equivalent to a single task of cost $W = \sum_{i=1}^n w_i$.*

Indeed, in the optimal solution, the n tasks are executed at the same speed, and they can be replaced by a single task of cost W , which is executed at the same speed and consumes the same amount of energy.

Finally, consider fork and join graphs. Let $V = \{T_1, \dots, T_n\}$. We consider either a fork graph $G = (V \cup \{T_0\}, E)$, with $E = \{(T_0, T_i), T_i \in V\}$, or a join graph $G = (V \cup \{T_0\}, E)$, with $E = \{(T_i, T_0), T_i \in V\}$. T_0 is either the source of the fork or the sink of the join.

Theorem 1 (fork and join graphs) *When G is a fork (resp. join) execution graph with $n + 1$ tasks T_0, T_1, \dots, T_n , the optimal solution to $\text{MINENERGY}(G, D)$ is the following:*

- the execution speed of the source (resp. sink) T_0 is $s_0 = \frac{(\sum_{i=1}^n w_i^3)^{\frac{1}{3}} + w_0}{D}$;
- for the other tasks T_i , $1 \leq i \leq n$, we have $s_i = s_0 \times \frac{w_i}{(\sum_{i=1}^n w_i^3)^{\frac{1}{3}}}$ if $s_0 \leq s_{\max}$.

Otherwise, T_0 should be executed at speed $s_0 = s_{\max}$, and the other speeds are $s_i = \frac{w_i}{D'}$, with $D' = D - \frac{w_0}{s_{\max}}$, if they do not exceed s_{\max} (Proposition 1 for independent tasks). Otherwise there is no solution.

If no speed exceeds s_{\max} , the corresponding energy consumption is

$$\text{minE}(G, D) = \frac{\left(\left(\sum_{i=1}^n w_i^3 \right)^{\frac{1}{3}} + w_0 \right)^3}{D^2}.$$

Corollary 2 (equivalent tasks for speed) *Consider a fork or join graph with tasks T_i , $0 \leq i \leq n$, and a deadline D , and assume that the speeds in the optimal solution*

<http://www.springer.com/978-1-4939-2091-4>

Handbook on Data Centers

Khan, S.U.; Zomaya, A.Y. (Eds.)

2015, XIII, 1334 p. 439 illus., 283 illus. in color.,

Hardcover

ISBN: 978-1-4939-2091-4