Brief Announcement: Reclaiming the Energy of a Schedule, Models and Algorithms
Guillaume Aupy, Anne Benoit, Fanny Dufossé, Yves Robert

To cite this version:

HAL Id: hal-00857268
https://inria.hal.science/hal-00857268
Submitted on 3 Sep 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Brief Announcement: Reclaiming the Energy of a Schedule, Models and Algorithms *

Guillaume Aupy, Anne Benoit, Fanny Dufossé and Yves Robert
LIP, École Normale Supérieure de Lyon, France

ABSTRACT
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. We study the complexity of the problem for different models: continuous speeds, discrete modes, distributed either arbitrarily or regularly, and VDD-hopping.

Categories and Subject Descriptors
F.2.2 [Analysis of algorithms and problem complexity]: Nonnumerical Algorithms and Problems—Sequencing and scheduling

General Terms
Performance, Theory, Algorithms.

Keywords
Energy models, complexity, bi-criteria optimization, algorithms, scheduling.

1. INTRODUCTION
The energy consumption of computational platforms has recently become a critical problem, both for economic and environmental reasons. Their power consumption is the sum of a static part (the cost for a processor to be turned on) and a dynamic part, which is a strictly convex function of the processor speed. More precisely, a processor running at speed $s$ dissipates $s^3$ watts [4, 5] per time-unit, hence consumes $s^3 \times t$ joules when operated during $t$ units of time.

Energy-aware scheduling aims at minimizing the energy consumed during the execution of the target application. Obviously, it makes sense only if it is coupled with some performance bound to achieve, otherwise, the optimal solution always is to run each processor at the slowest possible speed. In this paper, 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, for example for security reasons.

Optimization problem. Consider an application task graph $G = (V,E)$, with $n = |V|$ tasks, $V = \{T_1, T_2, \ldots, T_n\}$: $E$ 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 $G$ have been allocated onto a parallel platform made up of identical processors. The execution graph generated by this allocation is $G^* = (V,E^*)$, with an augmented set of edges $E^* \subseteq E$, and 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 this $\text{MinEnergy}(G,D)$ optimization problem in the simpler case where each task is executed at constant speed (valid for all models but the VDD-hopping one). 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.

\[\text{Minimize} \quad \sum_{i=1}^{n} s_i^3 \times d_i\]
subject to (i) $w_i = s_i \times d_i$ for each $T_i \in V$
(ii) $t_i + d_i \leq t_j$ for each $(T_i, T_j) \in E$
(iii) $t_i \leq D$ for each $T_i \in V$

We have $d_i = w_i/s_i$, hence a geometric problem in the non-negative variables $t_i$ and $1/s_i$, with linear constraints and objective function rewritten as $\sum_{i=1}^{n} (1/s_i)^{-2} \times w_i$. Energy models. In all models, when a processor operates at speed $s$ during $d$ time-units, the corresponding consumed energy is $s^3 \times d$, which is the dynamic part of the energy consumption [4, 5]. We do not take static energy into account, because all processors are up and alive during the whole execution. We now detail the possible speed values in each energy model, which should be added as a constraint in Equation (1).

Continuous: processors can have arbitrary speeds, from 0 to a maximum value $s_{max}$, and a processor can change its speed at any time during execution. This model is unrealistic but theoretically appealing [2].

Discrete: processors have a set of possible speed values, or modes, denoted as $s_1, \ldots, s_m$. There is no assumption on the range and distribution of these modes. The speed of a processor cannot change during the computation of a task, but it can change from task to task [7].
polynomial in the size of the instance and in values. 

size numbers which are good approximations of the optimal we can always solve the problem numerically and get fixed-

ested expressions of this form for trees). We do not know (we have the cubic root of the sum of cubes for forks, and numerical schemes exist. However, as illustrated on simple

gramming problem (see [3, Section 4.5]) for which efficient

mial time when

Otherwise, should be executed at speed , if they do

not exceed . Otherwise, there is no solution.

Theorem 2. MINERGY can be solved in polynomial time when is a tree, or a series-parallel graph (in the latter case, assuming ).

For arbitrary execution graphs, we have a geometric pro-

programming problem (see [3, Section 4.5]) for which efficient numerical schemes exist. However, as illustrated on simple fork graphs, the optimal speeds are not expected to be ra-
tional numbers but instead arbitrarily complex expressions (we have the cubic root of the sum of cubes for forks, and nested expressions of this form for trees). We do not know how to encode such numbers in polynomial size of the input (the rational task weights and the execution deadline). Still, we can always solve the problem numerically and get fixed-size numbers which are good approximations of the optimal values.

2.2 Discrete models

Theorem 3. With the Vdd-Hopping model, MINERGY can be solved in polynomial time (via linear pro-

2. RESULTS

All proofs, algorithms, and related work can be found in the companion research report [1].

2.1 The Continuous model

Theorem 1. When is a fork graph with integer tasks , where is the source, the optimal solution to MINERGY is to execute at speed

and at speed if . Otherwise, should be executed at speed , and the other speeds are , with , if they do not exceed . Otherwise, there is no solution.

Theorem 2. MINERGY can be solved in polynomial time when is a tree, or a series-parallel graph (in the latter case, assuming ).

For arbitrary execution graphs, we have a geometric pro-

programming problem (see [3, Section 4.5]) for which efficient numerical schemes exist. However, as illustrated on simple fork graphs, the optimal speeds are not expected to be ra-
tional numbers but instead arbitrarily complex expressions (we have the cubic root of the sum of cubes for forks, and nested expressions of this form for trees). We do not know how to encode such numbers in polynomial size of the input (the rational task weights and the execution deadline). Still, we can always solve the problem numerically and get fixed-size numbers which are good approximations of the optimal values.

2.2 Discrete models

Theorem 3. With the Vdd-Hopping model, MINERGY can be solved in polynomial time (via linear pro-

2. RESULTS

All proofs, algorithms, and related work can be found in the companion research report [1].

2.1 The Continuous model

Theorem 1. When is a fork graph with integer tasks , where is the source, the optimal solution to MINERGY is to execute at speed

and at speed if . Otherwise, should be executed at speed , and the other speeds are , with , if they do not exceed . Otherwise, there is no solution.

Theorem 2. MINERGY can be solved in polynomial time when is a tree, or a series-parallel graph (in the latter case, assuming ).

For arbitrary execution graphs, we have a geometric pro-

programming problem (see [3, Section 4.5]) for which efficient numerical schemes exist. However, as illustrated on simple fork graphs, the optimal speeds are not expected to be ra-
tional numbers but instead arbitrarily complex expressions (we have the cubic root of the sum of cubes for forks, and nested expressions of this form for trees). We do not know how to encode such numbers in polynomial size of the input (the rational task weights and the execution deadline). Still, we can always solve the problem numerically and get fixed-size numbers which are good approximations of the optimal values.

2.2 Discrete models

Theorem 3. With the Vdd-Hopping model, MINERGY can be solved in polynomial time (via linear pro-

2. RESULTS

All proofs, algorithms, and related work can be found in the companion research report [1].

2.1 The Continuous model

Theorem 1. When is a fork graph with integer tasks , where is the source, the optimal solution to MINERGY is to execute at speed

and at speed if . Otherwise, should be executed at speed , and the other speeds are , with , if they do not exceed . Otherwise, there is no solution.

Theorem 2. MINERGY can be solved in polynomial time when is a tree, or a series-parallel graph (in the latter case, assuming ).

For arbitrary execution graphs, we have a geometric pro-

programming problem (see [3, Section 4.5]) for which efficient numerical schemes exist. However, as illustrated on simple fork graphs, the optimal speeds are not expected to be ra-
tional numbers but instead arbitrarily complex expressions (we have the cubic root of the sum of cubes for forks, and nested expressions of this form for trees). We do not know how to encode such numbers in polynomial size of the input (the rational task weights and the execution deadline). Still, we can always solve the problem numerically and get fixed-size numbers which are good approximations of the optimal values.