HAL open science

List and shelf schedules for independent parallel tasks to minimize the energy consumption with discrete or continuous speeds

Anne Benoit, Louis-Claude Canon, Redouane Elghazi, Pierre-Cyrille Heam

To cite this version:

Anne Benoit, Louis-Claude Canon, Redouane Elghazi, Pierre-Cyrille Heam. List and shelf schedules for independent parallel tasks to minimize the energy consumption with discrete or continuous speeds. Journal of Parallel and Distributed Computing, 2022. hal-03920697

HAL Id: hal-03920697

https://inria.hal.science/hal-03920697v1

Submitted on 3 Jan 2023

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.

List and shelf schedules for independent parallel tasks to minimize the energy consumption with discrete or continuous speeds*

Anne Benoit[a], Louis-Claude Canon[b], Redouane Elghazi[b,*], Pierre-Cyrille Héam[b]

[a] *LIP, ENS Lyon, France*
[b] *FEMTO-ST Institute, Univ. Bourgogne Franche-Comté, CNRS, France*

Abstract

Scheduling independent tasks on a parallel platform is a widely-studied problem, in particular when the goal is to minimize the total execution time, or makespan ($P||C_{\max}$ problem in Graham's notations). Also, many applications do not consist of sequential tasks, but rather parallel tasks, either rigid, with a fixed degree of parallelism, or moldable, with a variable degree of parallelism (i.e., for which we can decide at the execution on how many processors they are executed). Furthermore, since the energy consumption of data centers is a growing concern, both from an environmental and economical point of view, minimizing the energy consumption of a schedule is a main challenge to be addressed. One can then decide, for each task, on how many processors it is executed, and at which speed the processors are operated, with the goal to minimize the total energy consumption. We further focus on co-schedules, where tasks are partitioned into shelves, and we prove that the problem of minimizing the energy consumption remains NP-complete when static energy is consumed during the whole duration of the application. We are however able to provide an optimal algorithm for the schedule within one shelf, i.e., for a set of tasks that start at the same time. Several approximation results are derived, both with discrete and continuous speed models, and extensive simulations are performed to show the performance of the proposed algorithms.

1. Introduction

We consider the problem of scheduling independent tasks. Even though this problem has already been widely studied, in particular when aiming to minimize the total execution time (or makespan) for sequential tasks, there remain avenues for improvement for variants of the problem. Using the Graham notations [15], the typical problem that is studied is $P||C_{\max}$, i.e., the goal is to minimize the makespan when scheduling independent sequential tasks on a set of identical processors. The decision version of this problem in its

*A shorter version of this paper appeared in the proceedings of SBAC-PAD'2021 under the title *Shelf schedules for independent moldable tasks to minimize the energy consumption.* This version contains revised proofs easier to follow, new theoretical results for continuous speeds, and additional simulation results to evaluate the performance of the algorithms. This work has been supported by the EIPHI Graduate School (contract ANR-17-EURE-0002).

*Corresponding author

simplest form is already NP-complete (it is indeed identical to 2-Partition [14] when considering two processors). However, several well-known heuristics lead to very good approximation algorithms, as the classical Longest Processing Time (LPT) heuristic, or even some PTAS or FPTAS algorithms [17].

The problem becomes more complicated when dealing with parallel tasks. Now, each task i is a parallel task that executes concurrently on p_i processors. The greedy list scheduling algorithm that gives priority to longest jobs is then known to be a 2-approximation when tasks are *rigid* (p_i is given and fixed) [13].

In order to ease the scheduling, it can be useful to group tasks by shelves (or batches, packs, levels, etc.), and then the shelves are scheduled one after the other. All the tasks in a same shelf start their execution at the same time, and the next shelf starts only when all tasks of the previous shelf are done. This is typically referred to as *shelf-scheduling* or *co-scheduling*. Of course, one may then waste time, due to idle resources if tasks do not all take the same time. However, such schedules are easy to implement and they also may have some theoretical guarantees. Indeed, the list scheduling that gives priority to longest jobs is known to be a 3-approximation when imposing the use of shelves [32] (recall that it is a 2-approximation without this restriction).

Such co-schedules are also very useful for *moldable* tasks, i.e., tasks whose degree of parallelism p_i can be chosen at execution. For such parallel moldable tasks, an easy way to proceed is to execute tasks sequentially, each task using the whole platform. However, it may be more efficient to group tasks by shelves, since the execution profile of a task may lead to less efficiency when using many processors. While the general problem is NP-hard, Aupy et al. [3] propose an optimal polynomial-time algorithm to decide the processor assignment that minimizes the makespan when there are at most two tasks in a shelf.

While most scheduling problems are focusing on makespan minimization, another core problem is the *energy consumption*. In order to optimize this energy consumption, modern processors can run at different speeds, and their power consumption is then 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. Indeed, the execution of a given amount of work costs more power if a processor runs at a higher speed [18]. More precisely, a processor running at speed s dissipates a power of s^3 Watts [19, 27, 10, 4, 11], hence it consumes an energy of $s^3 \times d$ Joules when operated during d units of time. Faster speeds allow for a faster execution, but they also lead to a much higher (supra-linear) power consumption. A more general model states that the power can be in s^α, where $2 \leq \alpha \leq 3$ [5]. While minimizing the makespan helps reducing the energy consumption, which increases with execution time, to the best of our knowledge, no study has been aiming at minimizing the energy consumption for shelf schedules.

For the static energy consumption, it depends on the time during which processors are powered. We consider two models: in the *independent* model, each processor is independently powered and can be turned off when not computing, hence the static power is paid only while processors are running. However, in the

simultaneous model, the platform is turned on as long as one processor is running, hence the static power must also be paid for idle processors.

Our main contributions are the following:

- We formalize the problem of scheduling independent moldable tasks to minimize energy consumption (MinE-Mold problem) with various model variants.
- We prove that the problem can be solved in polynomial time when processors are independently powered, while the problem becomes NP-complete with simultaneously powered processors.
- We establish multiple approximation ratios for both classical list scheduling algorithms, and shelf-based schedules, both with a realistic model where speeds can be chosen in a discrete set, and with the general model where speeds can take any positive real value (continuous speeds).
- We provide an optimal dynamic programming algorithm to minimize the energy consumption of a single shelf, both with the discrete and with the continuous model. The goal is to decide on how many processors to execute each task of the shelf, and at which speed to operate the task.
- We perform an empirical study and we show that, for most instances, a single speed can be used for all tasks without increasing the energy consumption. Also, as expected, shelf-based solutions consume more energy, but they are easier to implement and solutions are derived with a much lower complexity. A comparison with solutions using continuous speeds highlights that further energy savings could be achieved by carefully choosing the processor's speeds.

We first discuss related work in Section 2. Next, we detail the model (platform, tasks, energy consumption) and schedules, and we introduce the target optimization problems in Section 3. The complexity of the problems is established in Section 4. Approximation ratios for MinE-Mold are derived in Section 5 with discrete speeds and in Section 6 with continuous speeds. Optimal algorithms for a single shelf are provided in Section 7. Finally, a comprehensive empirical study is proposed in Section 8. We conclude and give hints for future research directions in Section 9.

2. Related work

Although the problem of minimizing the energy consumption of parallel platforms has been extensively studied, few works propose guaranteed scheduling algorithms for moldable tasks. We first cover approximation algorithms for the problem of minimizing the makespan because our approach relies on such results, even though we rather focus on minimizing the energy consumption. We then discuss heuristics proposed for real-time systems, which consider a model slightly different than ours. Finally, we present some examples of moldable task applications.

Makespan minimization. For the rigid case where the number of processors required by each task is fixed, classical *list scheduling algorithms*, denoted by ListBased, are widely used in the literature for

makespan minimization. Tasks are ordered in a priority list and are then scheduled by order of priority, as presented by Garey and Graham [13]: any time a processor is idle, the list is scanned in order and the first task that can be executed is started. Another way to say it, the principle of the algorithm is as follows: when resources are released, we see if a task can be started right now. If it is the case, we start it. If several tasks can be started, we take the one with the highest priority, given by an ordering of the tasks in a list. ListBased is a 2-approximation for the makespan. More precisely, it is a $2 \times \max(\frac{W}{p}, t_{\max})$-approximation, where $\frac{W}{p}$ is the average work and $t_{\max}$ is the execution time of the longest task.

Coffman et al. [12] made a landmark paper proving the approximation ratio of several *shelf-based algorithms* when considering rigid tasks only. They introduce the problem as a two-dimensional packing problem and focus on asymptotic performance bounds for the makespan. They show that the performance bounds of classic bin-packing heuristics Next-Fit Decreasing and First-Fit Decreasing are 3 and 2.7, respectively.

In [22], Krishnamurti et al. study a problem where processors are partitioned, and each task is submitted to one such partition with the objective to minimize the execution time. The number of partitions is bounded, which limits the maximum number of simultaneous tasks.

In [32], Turek et al. study the multi-shelves problem, still for makespan minimization. They first propose an allocation strategy for rigid tasks, and then use this strategy on several "allocation candidates" for the moldable case. This first strategy involves *co-schedules*, ShelfBased [32, LTF], where rigid tasks are partitioned into shelves, and all the tasks in a same shelf begin their execution at the same time (this is equivalent to Next-Fit Decreasing). Tasks are sorted in order of decreasing execution times. Then, tasks are inserted iteratively in the current shelf until the next task cannot be inserted. At this point, a new shelf is created and the process continues. A possible extension consists in allowing backfilling of previous shelves. ShelfBased is a 3-approximation for the makespan. More precisely, it is a $\left(\frac{2W}{p} + t_{\max}\right)$-approximation. To deal with moldable tasks, the authors also present an overall design [32, GF] that works as follows. First, a task and a number of processors are selected, and we assume that all tasks will complete before this one. Then, the number of processor is chosen for all other tasks so that their work is minimized. Finally, this instance is solved as if tasks were rigid (with a fixed number of processors. All pairs of initial task and number of processors are tried. Turek et al. also designed a 2.7-approximation for the multi-shelves problem, with a fixed number of shelves [31]. However, this algorithm is exponential in the number of shelves.

Aupy et al. go beyond the problem of minimizing the makespan, by tackling the problem of optimizing the power consumption, the makespan and the reliability [2]. However, they consider dependent non-parallel tasks, which is a setting completely different from ours, since we focus on parallel tasks. They show that most problems are NP-hard and propose heuristics.

Real-time systems. Finally, several works have been proposed in the context of real-time systems with moldable tasks, power constraints and deadlines. The closest to the problem that we target considers level-

based scheduling (similar to shelves) with rigid or moldable tasks [21]. They propose heuristics that extend bin-packing ones such as First-Fit Decreasing, Best-Fit Decreasing, etc. Most other works related to real-time systems propose heuristics [33, 34, 23]. In this paper, we do not consider deadlines and we investigate algorithms with guarantees.

Moldable Tasks. Most distributed algorithms have a time complexity that depends on the number of processors and therefore correspond to moldable tasks. This is the case for very classical algorithms such as the Fast Fourier Transform or the product of matrices by Strassen's method. For a more recent example, one can for instance point out distributed algorithms for generating samples from a large tabular [28]. In the context of biological applications, E. Saule et al. have shown how to use moldable tasks to tackle the short sequence mapping problem [29]. Other applicative examples of moldable tasks on stream algorithms are developed in [20].

Overall, we are not aware of any paper tackling directly the problem that we consider in this paper, namely the problem of scheduling independent moldable tasks to minimize the energy consumption, under different model variants (shelf-based solutions, discrete and continuous speeds, ...), and hence we were not able to directly compare our proposed approach to any algorithm coming from related work.

3. Model

We first describe the platform model (Section 3.1), the task model (Section 3.2), the energy model (Section 3.3), before formally defining general schedules, single-speed schedules and co-schedules in Section 3.4. Finally, we introduce the target optimization problems in Section 3.5. Table 1 summarizes the main notations used throughout the paper.

3.1. Platform

The target platform consists in p identical processors, whose frequency can be scaled using DVFS (Dynamic Voltage and Frequency Scaling).

These processors have a static power P_{stat} and a set $S = \{s_1, s_2, \ldots, s_k\}$ of possible speeds (or frequencies). For convenience, we let $s_{\min} = s_1$ and $s_{\max} = s_k$ be the minimum and maximum speeds. Indeed, current processors have a set of predefined speeds (or frequencies), which correspond to different voltages that the processor can be subjected to [25] (*discrete* model).

For the sake of completeness, we also consider the *continuous* model, where processors may be operated at any speed ($S = \mathbb{R}_+^*$). While this model is unrealistic (even though the number of available frequencies tends to be large in modern processors), it is theoretically appealing [6]. Also, a study of the problem without a constrained set of speeds allows for a better understanding of how to choose the available speeds during the design of a processor.

Notation	Quantity
p	Number of processors
P_{stat}	Static Power
S	Set of available speeds on the processors
T_i	Task number i
$w_{i,j}$	Total work for the execution of task i on j processors
$t_{i,j,s}$	Execution time of task i on j processors at speed s
$t_{i,j}$	Execution time of task i on j processors at speed $s = 1$
$a_{i,j,s}$	Area of task i on j processors at speed s
λ	A schedule
$p_i(\lambda)$	Number of processors allocated to task i in schedule λ
$s_i(\lambda)$	Speed of the processors allocated to task i in schedule λ
$t_{\max}(\lambda)$	Execution time of the longest task in schedule λ
$C_{\max}(\lambda)$	Makespan of schedule λ
$W(\lambda)$	Sum of the work of all tasks in schedule λ
$T_{dyn}(\lambda)$	Sum of the execution times of all tasks in schedule λ
$A_{dyn}(\lambda)$	Sum of the areas of all tasks in schedule λ
$A_{stat}(\lambda)$	Sum of the times during which processors are powered in schedule λ

Table 1: List of notations.

In our model, we don't consider the possibility of switching frequencies during the execution of a given task as it would never provide better solutions. The reason for that is that, due to the convexity of the function describing the energy consumption, taking the average speed of a task as its constant speed always incurs a lower energy consumption for the same execution time. Two different tasks, however, can be executed at different frequencies even if they are scheduled on a same processor. This remark would not necessarily remain true if we were to remove the full clairvoyance on the execution times of the tasks or if we were to add uncertainty to the energy model.

3.2. Tasks

We consider n moldable tasks $\{T_1, T_2, \ldots, T_n\}$ with respective execution profiles $(w_{i,j})_{i \in [\![1,n]\!], j \in [\![1,p]\!]}$, where $w_{i,j}$ is the total work required to execute T_i on j processors. The work is the total number of elementary operations to be executed by the processors. If executed at a speed of one, the time per processor is then $t_{i,j} = \frac{w_{i,j}}{j}$.

We assume that:

- $\forall i, (t_{i,j})_j$ is non-increasing in j (the more processors there are, the less time it will take per processor);
- $\forall i, (w_{i,j})_j$ is non-decreasing in j (when using more processors, there is more overhead due to the parallelization, which is a common assumption [9, 8]).

The algorithms presented in this paper do not require these two usual assumptions, however having them simply allows us to get better time complexities. For example, in the case where some allocations of

processors are not possible for some tasks, the corresponding processing time would be infinite and both assumptions could not hold.

Furthermore, for task T_i $(1 \leq i \leq n)$,

- p_i is the number of allocated processors;
- s_i is the speed of the processors during their execution;
- $t_{i,p_i,s_i} = \frac{t_{i,p_i}}{s_i} = \frac{w_{i,p_i}}{s_i \times p_i}$ is the execution time;
- $a_{i,p_i,s_i} = t_{i,p_i,s_i} \times p_i = \frac{w_{i,p_i}}{s_i}$ is the area of the rectangle representing this task.

3.3. Energy consumption

The energy consumption consists first of a static part, which corresponds to the power consumed when processors are turned on. The static power is denoted P_{stat}, and the corresponding static energy consumption on each processor is $t_{stat} \times P_{stat}$, where t_{stat} is the duration during which the processor is powered.

There is also a dynamic energy consumption, directly related to the speed s at which the processor operates, and the time t_{dyn} spent computing (which may be equal to or smaller than the time t_{stat}). Using a general model, the dynamic energy consumption is $t_{dyn} \times s^\alpha$ [5], where $\alpha > 1$ (in general, $2 \leq \alpha \leq 3$). Hence, for task T_i, the dynamic energy consumption on each processor is $t_{i,p_i,s_i} \times s_i^\alpha$ (since $t_{dyn} = t_{i,p_i,s_i}$), and the total dynamic energy consumption for the task is $a_{i,p_i,s_i} \times s_i^\alpha$ (the same energy is consumed by each of the p_i processors operating task T_i).

The case where $t_{stat} = t_{dyn}$ is the *independent* model, where each processor is independently powered, and hence turned off when it is not computing. We also consider the *simultaneous* model, where the whole platform remains powered as long as at least one processor is executing ($t_{stat} = C_{\max}$).

3.4. Schedules

Given a computational platform and a set of moldable tasks as described above, a schedule λ is a function that maps each task T_i to a tuple (M_i, s_i, δ_i), where M_i is the set of processors assigned to T_i (hence the number of processors assigned to the task is $p_i = |M_i|$), s_i is the speed of these processors to execute T_i, and $\delta_i \geq 0$ is the starting time of T_i. Moreover, λ must verify the following conditions:

- There exists i such that $\delta_i = 0$ (there is a task starting at time 0);
- If tasks T_i and $T_{i'}$ $(1 \leq i, i' \leq n$ and $i \neq i')$ are such that $M_i \cap M_{i'} \neq 0$, then $[\delta_i, \delta_i + t_{i,p_i,s_i}] \cap [\delta_{i'}, \delta_{i'} + t_{i',p_{i'},s_{i'}}] = \emptyset$ (a processor cannot be used for two different tasks at the same time).

We also have the following aggregated quantities depending on a schedule λ:

- $C_{\max}(\lambda) = \max_i\{\delta_i + t_{i,p_i,s_i}\}$ is the makespan (or total execution time);

- $W(\lambda) = \sum_{i=1}^{n} w_{i,p_i}$ is the cumulative work;
- $T_{dyn}(\lambda) = \sum_{i=1}^{n} t_{i,p_i,s_i}$ is the cumulative execution time of all tasks;
- $A_{dyn}(\lambda) = \sum_{i=1}^{n} a_{i,p_i,s_i}$ is the cumulative execution time on all processors;
- $A_{stat}(\lambda)$ is the cumulative time on all processors during which they are powered. It is either $A_{dyn}(\lambda)$ in the *independent* model, or it is $p \times C_{\max}(\lambda)$ in the *simultaneous* model.

We can then express the total energy consumption of a schedule λ as:

$$E(\lambda) = \sum_{i=1}^{n} a_{i,p_i,s_i} \times s_i^{\alpha} + A_{stat}(\lambda) \times P_{stat}.$$

If there is no ambiguity on λ, we write $C_{\max}$ for $C_{\max}(\lambda)$; and similarly for related quantities (W, T_{dyn}, A_{dyn}, etc.).

Finally, we pay a particular attention to two classes of particular schedules:

- *Single-speed* schedules are schedules such that all the speeds are equal for all tasks, i.e., for $1 \leq i \leq n$, $s_i = s \in S$.
- *Co-schedules* are organized as shelves, as motivated in Section 1. A co-schedule consists of a partition of the tasks into shelves and such that:
  - If two tasks are in the same shelf, then they start their execution at the same time;
  - If two tasks are not in the same shelf, then one finishes its execution before the other one starts.

3.5. Optimization problems

The general problem is MinE-Mold: Given n moldable tasks, their execution profiles $(w_{i,j})$, p processors and their speeds S, the goal is to find a schedule that minimizes the total energy consumption. We focus mainly on the case with a set of discrete speeds (*discrete* model) and simultaneously-powered processors (*simultaneous* model). Variants with continuous speeds and/or independently-powered processors are referred to by adding Cont or Indep to the problem name (hence, MinE-Mold-Cont-Indep is the problem with both variants, while MinE-Mold-Indep is the problem with discrete speeds and independently-powered processors).

We also consider the variant of the problem with *rigid* tasks, i.e., when the speed s_i and the number of processors per task p_i are fixed (MinE-Rig problem), again by default with discrete speeds and the *simultaneous* model.

Moreover, we add SS to refer to the variant of any problem where the same speed must be selected for each task (see *single-speed* schedules above).

Problem	Processors per task	Speeds	Static area A_{stat}	Constraint
MinE-Mold-Indep	Variable $p_i \in \mathbb{N}$	Finite set S	$A_{stat} = A_{dyn}$	$\emptyset$
MinE-Mold	Variable $p_i \in \mathbb{N}$	Finite set S	$A_{stat} = p \times C_{\max}$	$\emptyset$
MinE-Rig	Fixed $p_i \in \mathbb{N}$	Finite set S	$A_{stat} = p \times C_{\max}$	$\emptyset$
MinE-Mold-Cont	Variable $p_i \in \mathbb{N}$	Interval	$A_{stat} = p \times C_{\max}$	$\emptyset$
MinE-Rig-Cont	Fixed $p_i \in \mathbb{N}$	Interval	$A_{stat} = p \times C_{\max}$	$\emptyset$
MinE-OneShelf	Variable $p_i \in \mathbb{N}$	Finite set S	$A_{stat} = p \times C_{\max}$	$\sum_{i=1}^{n} p_i \leq p$

Table 2: List of problems (multiple speeds).

Finally, since we are interested in co-schedules, we consider the more constrained problem with *a single shelf*, i.e., all tasks must start at time 0 and be executed concurrently. The corresponding problem is MinE-OneShelf: Given a set of n tasks and p processors, the goal is to minimize the energy consumption knowing that all tasks start at time 0 ($\delta_i = 0$ for $1 \leq i \leq n$). Solving this particular problem will help us derive efficient co-schedules for the general MinE-Mold problem. More precisely, a solution to MinE-OneShelf is an assignment $((p_i)_{i \in [\![1,n]\!]}, (s_i)_{i \in [\![1,n]\!]})$ such that:

- $\forall i \in [\![1, n]\!]$, task T_i is executed on $p_i \geq 1$ processors at speed $s_i \in S$;
- $\sum_{i=1}^{n} p_i \leq p$ (at most p processors are used, since all tasks execute concurrently).

All problems (with the multiple speed variant) are summarized in Table 2.

4. Problem complexity

We start with the study of the *independent* model, and we derive that MinE-Mold-Indep can be solved in polynomial time (Section 4.1). However, with the more realistic *simultaneous* model, we prove that MinE-Mold is NP-complete (Section 4.2).

4.1. Optimal algorithm for MinE-Mold-Indep

In the *independent* model, the platform has multiple nodes that are independently powered, which means that each node can individually be turned down at any point of the execution, and not consume any more energy. Therefore, the total energy consumption is the sum of the individual energy consumption of each task. Hence, for each task, we need to decide on how many processors it should be executed, and at which speed, in order to minimize its energy consumption. Recall that we solely focus on energy optimization, and hence we do not have any constraint on the total time to completion.

Since the $(w_{i,j})$'s are non-decreasing in j, we have $a_{i,1,s_i} \leq a_{i,p_i,s_i}$ for all $1 \leq p_i \leq p$. Therefore, a_{i,p_i,s_i} is minimized if a single processor is used (independently of the speed that is chosen). Hence, we set $p_i = 1$, i.e., the task is executed on a single processor (its execution time may be long, but other processors will be turned off and less energy will be consumed, since we have independently-powered processors).

We still need to decide at which speed to execute the task on its single processor. Indeed, there is a tradeoff between executing the task fast to reduce the static energy consumption of the task (the area $a_{i,1,s_i}$, that decreases with s_i), and running at a slower speed to reduce the dynamic energy consumption, which is in s_i^α and hence increases with s_i. The total energy consumption of the task is $a_{i,1,s_i} \times (s_i^\alpha + P_{stat})$, and $a_{i,1,s_i} = \frac{w_{i,1}}{s_i}$. The goal is therefore to find s_i that minimizes $s_i^{\alpha-1} + \frac{P_{stat}}{s_i}$.

The optimal value of s_i (denoted by s_i^{opt}) can then be easily found in $O(|S|)$ if S is a set of discrete speeds, by comparing the values for every possible speed $s_i \in S$. In the continuous case, we remark that the function $f : s \mapsto s_i^{\alpha-1} + \frac{P_{stat}}{s_i}$ is a convex function that reaches its minimum at $s_i^{opt} = \sqrt[\alpha]{\frac{P_{stat}}{\alpha-1}}$, and hence it can be found in $O(1)$.

We can therefore optimize the energy consumption of each task independently, and execute the tasks one after the other. For each task, as shown above, we execute task i on a single processor at speed s_i^{opt}. Again, we focus solely on energy optimization, and the time to completion might be very large in this case (a single processor is used). Of course, one can also schedule the tasks on different processors and achieve the same energy consumption with a smaller total execution time since nodes are independently powered. Anyway, the optimal solution to this problem can therefore be found in polynomial time.

In the rest of this paper, unless otherwise stated, we focus on simultaneously powered processors (i.e., $A_{stat} = p \times C_{\max}$). It then becomes crucial to use the whole platform and minimize the execution time, since the platform remains powered during the whole execution.

4.2. NP-completeness of MinE-Mold

When moving to the *simultaneous* model, it becomes crucial to also minimize the total execution time, since the static energy is consumed during the whole execution. We show that the MinE-Mold problem actually is NP-complete, even when a single speed is available. For the continuous case (MinE-Mold-Cont), the problem is also NP-hard, even though we do not know whether it is in NP or not because of the speeds in $\mathbb{R}_+^*$.

Theorem 1. *The decision problems associated to* MinE-Mold *and* MinE-Mold-SS *are NP-complete, and the decision problems associated to* MinE-Mold-Cont *and* MinE-Mold-Cont-SS *are NP-hard.*

Proof. We first prove that the decision problem associated to MinE-Mold is in NP: a certificate is a schedule, i.e., the number of processors and the speed of each task, as well as the starting time of each task, and it is easy to check in polynomial time whether the bound on energy consumption is achieved. However, in the continuous case, the speeds and starting time of tasks might not be in $\mathbb{Q}$, and hence we do not know whether MinE-Mold-Cont is in NP or not.

To prove the NP-hardness, we do a reduction from the problem of 3-Partition [14]: Given $3n$ integers $\{a_1, \ldots, a_{3n}\}$ whose sum is $nB = \sum_{i=1}^{3n} a_i$ and with $\frac{B}{4} < a_i < \frac{B}{2}$ for $1 \leq i \leq 3n$, does there exist a partition

of $\{1, \ldots, 3n\}$ into n subsets $S_1, \ldots, S_n$, such that $\sum_{i \in S_j} a_i = B$ for $1 \leq j \leq n$?

Let $\mathcal{I}_1$ be an instance of 3-Partition. We create an instance $\mathcal{I}_2$ of MinE-Mold (or MinE-Mold-Cont) with n processors, and $3n$ tasks that cannot be parallelized, i.e., their execution time is not improved when using more than one processor. Hence, task i ($1 \leq i \leq 3n$) is such that $t_{i,j} = a_i$ for $1 \leq j \leq n$. Furthermore, we have $P_{stat} = 2$ and $\alpha = 3$. In the discrete version of the problem, there is a single speed $S = \{1\}$. Finally, we set the bound on total energy consumption for $\mathcal{I}_2$ to $3nB$.

First, note that it is always better to execute each task on a single processor. Indeed, if a task is executed on more than one processor, it takes the same time to execute but consumes additional energy. Second, in the continuous case, if a single task is considered, it should be executed at speed $s_i^{opt} = \sqrt[\alpha]{\frac{P_{stat}}{\alpha - 1}}$ as shown in Section 4.1 for the independent model, which corresponds to a speed of one since $P_{stat} = 2$ and $\alpha = 3$. The use of another speed leads to a higher energy consumption for this task. Hence, assuming that each task is executed at speed 1 (both in the discrete and continuous cases), we obtain a dynamic energy consumption of a_i for task i, and a total dynamic energy consumption of nB. The static energy consumption depends on the total execution time t, and it is $P_{stat} \times t \times n$. If there is no idle time, and hence no waste of static energy, the time $t \times n$ also corresponds to the total time spent executing the tasks as in the independent model, which is nB as for the dynamic energy consumption. We are now ready to prove the equivalence of solutions.

If $\mathcal{I}_1$ has a solution, we execute tasks of a same subset S_j onto processor j, for $1 \leq j \leq n$. Each processor completes in time B, and the static energy consumption is $2nB$, hence a total energy consumption of $3nB$ (static energy plus dynamic energy). Therefore, $\mathcal{I}_2$ has a solution.

If $\mathcal{I}_2$ has a solution, we define S_j as the set of tasks executed on processor j. Since the energy consumption is not greater than $3nB$, each task must be executed at speed 1 on a single processor, otherwise the sum of the energy consumption of each task (as in the independent model) would exceed $3nB$, and lead to a contradiction. Indeed, the energy consumption with the simultaneous model is at least as high as the one with the independent model as it may account for extra static energy consumption due to some idle time of processors. Given that each task is executed at speed 1, the total dynamic energy consumption is nB and the static energy consumption cannot exceed $2nB$. This means that the total execution time must be such that $t \leq B$, and the sum of the a_i's in each subset S_j cannot exceed B. Therefore, $\mathcal{I}_1$ has a solution, which concludes the proof. □

5. Approximation ratios with discrete speeds

To solve MinE-Mold, we extend a strategy [32, GF] that transforms a moldable instance into multiple rigid ones by fixing the number of processors (and possibly the speed) of each task. Then, each rigid instance is solved with a heuristic, for example the ones that we mentioned earlier, ListBased (the classical list-based

scheduling where no processor is left idle if a task can be started, which is a 2-approximation algorithm for the total execution time, or makespan) or ShelfBased (rigid tasks are partitioned into shelves, and all the tasks in a same shelf begin their execution at the same time, which is a 3-approximation algorithm for makespan).

We thus start by considering the rigid case (MinE-Rig problem) and derive approximation results for energy consumption. Moreover, we first consider a simplified version of the problem where all tasks have the same speed (Section 5.1), before moving to the general case (Section 5.2). By convention, λ^{OPT} refers to the optimal schedule that minimizes the energy consumption and $\lambda_\bullet$ to the schedule at speed $s_\bullet$ with a guaranteed bound on the energy consumption. Moreover, for a rigid instance, λ^* is the schedule with minimum makespan and $\lambda_{\mathcal{A}}$ a schedule with a guaranteed bound on the makespan.

5.1. Processors with a single speed ($s_i = s$)

We first consider the case where the speed must be the same for all tasks, i.e., $s_i = s$ for $1 \leq i \leq n$. The energy simplifies as:

$$E = \sum_{i=1}^{n} a_{i,p_i,s_i} \times s^\alpha + A_{stat} \times P_{stat},$$

with $\alpha > 1$.

5.1.1. Rigid case

We start with the rigid case, which means that, for $1 \leq i \leq n$, the number of processors p_i for task i is fixed. Hence, the workload for task i is also known (w_{i,p_i}). Moreover, for a given schedule λ, all the s_i's are equal to s_λ. The tuple $\lambda(i)$ is hence denoted as $(M_i, s_\lambda, \delta_i)$.

Given any $\rho > 0$, we denote by $\rho\lambda$ the schedule associating to each task i the tuple $(M_i, \rho \times s_\lambda, \frac{\delta_i}{\rho})$, i.e., the speed is scaled by a factor ρ, and the starting times are adjusted accordingly, without any modification in the processor allocation. One can easily check that $\rho\lambda$ is also a rigid single-speed schedule.

Two schedules λ_1 and λ_2 are equivalent, denoted $\lambda_1 \sim \lambda_2$, if there exists $\rho > 0$ such that $\lambda_1 = \rho\lambda_2$. The relation $\sim$ is an equivalence relation. The equivalence class of λ is denoted $[\lambda]$.

Recall that $C_{\max}(\lambda) = \frac{A_{stat}(\lambda)}{p}$ is the makespan: it is the total duration during which the whole system is powered. For convenience, we define the following quantity:

$$K_{[\lambda]} = s_\lambda \times C_{\max}(\lambda).$$

It is easy to see that this is a constant for the equivalence class of λ; indeed, given any $\rho > 0$, $C_{\max}(\rho\lambda) = \frac{C_{\max}(\lambda)}{\rho}$, and $s_{\rho\lambda} = \rho \times s_\lambda$. This is used as makespan that is normalized to the speed.

The goal of this section is to prove the following theorem. The idea consists in considering algorithms with a given approximation ratio on the makespan and show how these ratios extend to the energy minimization.

In particular, we consider the same allocation as the one returned by the approximation algorithm but with a speed that minimizes the energy consumption.

Theorem 2. *In the rigid single-speed context (MinE-Rig-SS) and assuming that there exists an algorithm $\mathcal{A}$ that yields a c-approximation of the optimal makespan, we can compute in polynomial time a schedule consuming at most c times the optimal energy.*

Proof. In the rigid case, we have $\sum_{i=1}^{n} a_{i,p_i,s_i} = \sum_{i=1}^{n} \frac{w_{i,p_i}}{s_i}$. The cumulative work $W = \sum_{i=1}^{n} w_{i,p_i}$ is independent of the schedule λ. We can then write the energy consumption of λ as:

$$E(\lambda) = \frac{W}{s_\lambda} \times s_\lambda^\alpha + A_{stat}(\lambda) \times P_{stat} \tag{1}$$

$$= W \times s_\lambda^{\alpha-1} + p \times C_{\max}(\lambda) \times P_{stat}. \tag{2}$$

Let λ^{OPT} be a single-speed schedule minimizing the energy consumption. Let us denote by $\lambda_{\mathcal{A}}$ the schedule returned by $\mathcal{A}$. Let $s_\bullet \in S$ be a speed for which $\min_{\lambda \in [\lambda_{\mathcal{A}}]} E(\lambda)$ is attained. We analyze the schedule $\lambda_\bullet$ defined by the same allocation as $\mathcal{A}$, but with the speed $s_\bullet$. Its makespan is $C_{\max}(\lambda) = \frac{s_{\lambda_{\mathcal{A}}}}{s_\bullet} \times C_{\max}(\lambda_{\mathcal{A}})$.

Note that if λ^* is a single-speed schedule minimizing the makespan, then $C_{\max}(\lambda_{\mathcal{A}}) \leq c \times C_{\max}(\lambda^*)$ because $\mathcal{A}$ yields a c-approximation of the optimal makespan. Therefore, $K_{[\lambda_{\mathcal{A}}]} \leq c \times K_{[\lambda^*]}$ (necessarily $s_{\lambda_{\mathcal{A}}} \leq s_{\lambda^*} = s_{\max}$). Moreover, for any single-speed schedule λ, $K_{[\lambda^*]} \leq K_{[\lambda]}$, otherwise by running λ at speed $s_{\max}$, we would get a schedule with a lower makespan than λ^*, which would contradicts the fact that λ^* is optimal for the makespan..

$$\begin{aligned}
\min_{\lambda \in [\lambda_{\mathcal{A}}]} E(\lambda_\bullet) &= W \times s_\bullet^{\alpha-1} + \frac{s_{\lambda_{\mathcal{A}}}}{s_\bullet} p \times C_{\max}(\lambda_{\mathcal{A}}) \times P_{stat} && \text{Equation (2)} \\
&= W \times s_\bullet^{\alpha-1} + p \times \frac{K_{[\lambda_{\mathcal{A}}]}}{s_\bullet} \times P_{stat} && \text{definition of } K_{[\lambda_{\mathcal{A}}]} \\
&\leq W \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + p \times \frac{K_{[\lambda_{\mathcal{A}}]}}{s_{\lambda^{\text{OPT}}}} \times P_{stat} && \text{optimality of } s^*_{[\lambda_{\mathcal{A}}]} \\
&\leq W \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + c \times p \times \frac{K_{[\lambda^*]}}{s_{\lambda^{\text{OPT}}}} \times P_{stat} && K_{[\lambda_{\mathcal{A}}]} \leq c \times K_{[\lambda^*]} \\
&\leq W \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + c \times p \times \frac{K_{[\lambda^{\text{OPT}}]}}{s_{\lambda^{\text{OPT}}}} \times P_{stat} && K_{[\lambda^*]} \leq K_{[\lambda^{\text{OPT}}]} \\
&\leq W \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + c \times p \times C_{\max}(\lambda^{\text{OPT}}) \times P_{stat} && \text{definition of } K_{[\lambda^{\text{OPT}}]} \\
&\leq c \times W s_{\lambda^{\text{OPT}}}^{\alpha-1} + c \times p \times C_{\max}(\lambda^{\text{OPT}}) \times P_{stat} && c \geq 1 \\
&\leq c \times E(\lambda^{\text{OPT}}), && \text{Equation (2)}
\end{aligned}$$

thus proving the theorem. □

5.1.2. Moldable case

The algorithm for MinE-Mold-SS (Algorithm 1) assumes first that a speed of one is used. First, we select both a task $T_{i'}$ and the number of processors $p_{i'}$ for this task. Let $t_{\max}$ be the longest execution time among all tasks (assuming a speed of one). We assume that this time is achieved with this task (i.e., $t_{\max} = t_{i',p_{i'}}$). There are np such selections, and we explore them all. For each value of $t_{\max}$ (i.e., for each pair $(T_{i'}, p_{i'})$), we select the number of processors of each other task to be associated with the lowest work such that $t_{i,p_i} \leq t_{i',p_{i'}}$ still holds. We then solve the rigid instance obtained by fixing the number of processors for each task with ListBased-SS or ShelfBased-SS, and we select the speed that minimizes the energy for the resulting schedule. The final schedule is the one with minimum energy over the np explored possibilities.

Algorithm 1: Algorithm for MinE-Mold-SS

for $(T_{i'}, p') \in \{T_1, \ldots, T_n\} \times \{1, \ldots, p\}$ **do**
  $t_{\max} \leftarrow t_{i',p'}$;
  for $T_i \in \{T_1, \ldots, T_n\}$ **do**
    $p_i \leftarrow \arg\min_{1 \leq j \leq p} j \times t_{i,j}$ such that $t_{i,p_i} \leq t_{\max}$ if it exists;
    $p_i = 0$ otherwise;
  if *all* $p_i \neq 0$ **then**
    Apply a guaranteed algorithm $\mathcal{A}$ on the rigid instance $\{(T_1, p_1), \ldots, (T_n, p_n)\}$ at speed of 1, to get a schedule $\lambda_\bullet^{(T_{i'},p')}$;
    Select the speed $s_\bullet$ that minimizes the energy ;
return the schedule $\lambda_\bullet$ with minimum energy among all the computed $\lambda_\bullet^{(T_{i'},p')}$;

Intuitively, we analyze the approximation ratio of any moldable scheduling algorithm with the following approach based on [32, GF]:

- For a given $t_{\max}$, we bound the cumulative work to be executed assuming any task execution duration is bounded by $t_{\max}$.
- We then bound the maximum makespan achievable with a guaranteed algorithm for MinE-Rig-SS.
- Finally, we bound the maximum total energy consumption during this duration.

We can state the main result of this section.

Theorem 3. *We assume that there exists a polynomial-time algorithm $\mathcal{A}$ for MinE-Rig-SS that returns a schedule $\lambda_\mathcal{A}$ at a speed of one such that $C_{\max}(\lambda_\mathcal{A}) \leq a \times \frac{W(\lambda_\mathcal{A})}{p} + b \times t_{\max}(\lambda_\mathcal{A})$ (resp. $C_{\max}(\lambda_\mathcal{A}) \leq \max\left(a \times \frac{W(\lambda_\mathcal{A})}{p}, b \times t_{\max}(\lambda_\mathcal{A})\right)$). In the moldable single-speed context (MinE-Mold-SS), one can compute in polynomial time a schedule $\lambda_\bullet$ that consumes at most $a + b$ (resp. $\max(a, b)$) times the optimal energy.*

Proof. The proof is done for $C_{\max}(\lambda_\mathcal{A}) \leq a \times \frac{W(\lambda_\mathcal{A})}{p} + b \times t_{\max}(\lambda_\mathcal{A})$. The other case (maximum of the two terms instead of sum) is similar.

For any task i and any number p_i of processors, we denote by λ_{i,p_i} the schedule returned by $\mathcal{A}$ on the following rigid instance: for all i', $p_{i'}$ is the integer in $\{1,\ldots,p\}$ minimizing $w_{i',p_{i'}}$ under the constraint $t_{i',p_{i'}} \leq t_{i,p_i}$. There are at most np different such schedules and each one can be computed in polynomial time. For each schedule, the selected speed is the one that minimizes the energy consumption. Let $\lambda_\bullet$ be a schedule of $(\lambda_{i,p_i})_{i,p_i}$ (where each task is running at speed s_{i,p_i}) with minimum energy consumption (i.e., the schedule for which $E(\lambda_\bullet) = \min_{i,p_i} E(\lambda_{i,p_i})$).

Let λ^{OPT} be a schedule minimizing the energy (for MinE-Mold-SS). Let i^{OPT} denote the longest task in the optimal schedule λ^{OPT} and $p_{i^{\text{OPT}}}$ denote the number of processors for this longest task (i.e., $t_{\max}(\lambda^{\text{OPT}}) = t_{i^{\text{OPT}},p_{i^{\text{OPT}}}}$). By construction of the λ_{i,p_i}, one has $W(\lambda_g) \leq W(\lambda^{\text{OPT}})$ where $\lambda_g = \lambda_{i^{\text{OPT}},p_{i^{\text{OPT}}}}$. By definition, $t_{\max}(\lambda_g) = t_{\max}(\lambda^{\text{OPT}})$. Thus, at a speed of one, $K_{[\lambda_g]} = C_{\max}(\lambda_g) \leq a \times \frac{W(\lambda_g)}{p} + b \times t_{\max}(\lambda_g) \leq a \times \frac{W(\lambda^{\text{OPT}})}{p} + b \times t_{\max}(\lambda^{\text{OPT}}) \leq (a+b) \times C_{\max}(\lambda^{\text{OPT}}) = (a+b) \times K_{[\lambda^{\text{OPT}}]}$. Finally, remark that a and b are necessarily constants satisfying $a + b \geq 1$ because $\mathcal{A}$ would provide a schedule better than the optimal otherwise.

We have:

$$
\begin{aligned}
E(\lambda_\bullet) &\leq E(\lambda_g) && \text{optimality of } \lambda_\bullet \text{ over all } (\lambda_{i,p_i})_{i,p_i} \\
&\leq E(\frac{s_{\lambda^{\text{OPT}}}}{s_{\lambda_g}}\lambda_g) && \text{optimality of } s_g \\
&\leq W(\lambda_g) \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + p \times \frac{K_{[\lambda_g]}}{s_{\lambda^{\text{OPT}}}} \times P_{stat} && \text{Equation (2)} \\
&\leq W(\lambda^{\text{OPT}}) \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + (a+b) \times p \times \frac{K_{[\lambda^{\text{OPT}}]}}{s_{\lambda^{\text{OPT}}}} \times P_{stat} && \text{approximation ratio of } \mathcal{A} \\
&\leq (a+b) \times W(\lambda^{\text{OPT}}) \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + (a+b) \times p \times \frac{K_{[\lambda^{\text{OPT}}]}}{s_{\lambda^{\text{OPT}}}} \times P_{stat} && a+b \geq 1 \\
&\leq (a+b) \times E(\lambda^{\text{OPT}}) && \text{Equation (2)},
\end{aligned}
$$

which concludes the proof. □

It has already been proved that ListBased-SS is an algorithm that outputs a schedule λ at a speed of one such that $C_{\max}(\lambda) \leq \max\left(2 \times \frac{W(\lambda)}{p}, 2 \times t_{\max}(\lambda)\right)$ [13], so by applying Theorem 3 with a maximum and $a = 2$ and $b = 2$, we show that ListBased-SS is a 2-approximation algorithm for the energy.

As for ShelfBased-SS, it is an algorithm that outputs a schedule λ such that $C_{\max}(\lambda) \leq 2 \times \frac{W(\lambda)}{p} + t_{\max}(\lambda)$ [32, LTF], so by applying Theorem 3 with a sum and $a = 2$ and $b = 1$ [32, LTF], we show that ShelfBased-SS is thus a 3-approximation algorithm for the energy.

5.2. Processors with different speeds for each task

When generalizing to multiple speeds, the approach is close to the one used for the single-speed problem where all tasks are executed at the same speed (see Algorithm 1); the corresponding algorithm is detailed

in Algorithm 2. Note that in the case where the speeds of the tasks are already determined, the dynamic area A_{dyn} is equivalent to the work from [31], which was denoted by W in Theorem 3.

Algorithm 2: Algorithm for MinE-Mold, with multiple speeds

```
for (T_{i'}, p', s') ∈ {T_1, ..., T_n} × {1, ..., p} × {s_1, ..., s_k} do
    t_max ← t_{i',p',s'} ;
    for T_i ∈ {T_1, ..., T_n} do
        p_i, s_i ← arg min_{1≤j≤p, s∈S} a_{i,j,s} × s^α + a_{i,j,s} × P_stat such that t_{i,p_i,s_i} ≤ t_max if it exists;
        p_i, s_i ← 0, 0 otherwise.
    if all p_i, s_i ≠ 0, 0 then
        Apply a guaranteed algorithm A on the rigid instance {(T_1, p_1, s_1), ..., (T_n, p_n, s_n)} at speed
         of 1, to get a schedule λ_•^{(T_{i'}, p', s')};
return the schedule λ_• with minimum energy among all the computed λ_•^{(T_{i'}, p_{i'}, s_{i'})} ;
```

Theorem 4. *We assume that there exists a polynomial-time algorithm $\mathcal{A}$ for MinE-Rig-SS that returns a schedule $\lambda_{\mathcal{A}}$ such that $C_{\max}(\lambda_{\mathcal{A}}) \leq a \times \frac{A_{dyn}(\lambda_{\mathcal{A}})}{p} + b \times t_{\max}(\lambda_{\mathcal{A}})$ (resp. $C_{\max}(\lambda_{\mathcal{A}}) \leq \max\left(a \times \frac{A_{dyn}(\lambda_{\mathcal{A}})}{p}, b \times t_{\max}(\lambda_{\mathcal{A}})\right)$) with $1 \leq a$. In the general moldable context (MinE-Mold), one can compute in polynomial time a schedule $\lambda_\bullet$ that consumes at most $a + b$ (resp. $\max(a, b+1)$) times the optimal energy.*

Proof. We consider in this proof the max case for $\mathcal{A}$. The proof is similar for the sum.

For any task i, any number p_i of processors and any speed s_i, we consider the following rigid instance: for all i', we select $(p_{i'}, s_{i'}) \in \{1, \ldots, p\} \times S$ that minimizes the energy consumption of task i, $a_{i',p_{i'},s_{i'}} \times s_{i'}^{\alpha} + a_{i',p_{i'},s_{i'}} \times P_{stat}$ where $a_{i',p_{i'},s_{i'}}$ is the area of the rectangle representing task i, under the constraint $t_{i',p_{i'},s_{i'}} \leq t_{i,p',s}$. The schedule returned by $\mathcal{A}$ for this problem is denoted λ_{i,p',s_i}. There are at most $np|S|$ different such schedules and each one can be computed in polynomial time. Let $\lambda_\bullet$ be a schedule among the $(\lambda_{i,p_i,s_i})_{i,p_i,s_i}$ minimizing the energy.

Let λ^{OPT} be a schedule minimizing the energy (for MinE-Mold). Let i^{OPT}, $p_{i^{\text{OPT}}}$ and $s_{i^{\text{OPT}}}$ satisfy $t_{\max}(\lambda^{\text{OPT}}) = t_{i^{\text{OPT}},p_{i^{\text{OPT}}},s_{i^{\text{OPT}}}}$. For a schedule λ, set $E_i(\lambda) = a_{i,p_i,s_i} s_i^{\alpha} + a_{i,p_i,s_i} P_{stat}$. By construction, one has:

$$\sum_i E_i(\lambda_{i^{\text{OPT}},p_{i^{\text{OPT}}},s_{i^{\text{OPT}}}}) \leq \sum_i E_i(\lambda^{\text{OPT}}). \tag{3}$$

To simplify the notation, we denote $\lambda_{i^{\text{OPT}},p_{i^{\text{OPT}}},s_{i^{\text{OPT}}}}$ by λ_g. Now,

$$\begin{aligned}
E(\lambda_\bullet) &\leq E(\lambda_{i^{\text{OPT}},p_{i^{\text{OPT}}},s_{i^{\text{OPT}}}}) = E(\lambda_g) \\
&\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^{\alpha} + A_{stat}(\lambda_g) \times P_{stat} && \text{definition of the energy} \\
&\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^{\alpha} + p \times C_{\max}(\lambda_g) \times P_{stat} && \text{definition of } A_{stat}
\end{aligned}$$

$$\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + p \times \max\left(a \times \frac{A_{dyn}(\lambda_g)}{p}, b \times t_{\max}(\lambda_g)\right) \times P_{stat} \quad \text{approximation ratio of } \mathcal{A}$$

Now, by distributivity, we have one of the two following possibilities:

$$E(\lambda_\bullet) \leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + a \times A_{dyn}(\lambda_g) \times P_{stat} \qquad \text{left-hand side of the max}$$

or

$$E(\lambda_\bullet) \leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + b \times p \times t_{\max}(\lambda_g) \times P_{stat} \qquad \text{right-hand side of the max}$$

We start with the left-hand side of the max:

$$\begin{aligned}
\text{LHS} &\triangleq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + a \times A_{dyn}(\lambda_g) \times P_{stat} \\
&\leq a \times (\sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + A_{dyn}(\lambda_g) \times P_{stat}) && 1 \leq a \\
&\leq a \times \sum_i (a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + a_{i,p_i,s_i}(\lambda_g) \times P_{stat}) && \text{definition of } A_{dyn} \\
&\leq a \times \sum_i E_i(\lambda_g) && \text{definition of } E_i \\
&\leq a \times \sum_i E_i(\lambda^{\text{OPT}}) && \text{Equation (3)} \\
&\leq a \times E(\lambda^{\text{OPT}}) && \sum E_i \leq E
\end{aligned}$$

Now, the right-hand side of the max:

$$\begin{aligned}
\text{RHS} &\triangleq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + b \times p \times t_{\max}(\lambda_g) \times P_{stat} \\
&\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + b \times p \times t_{\max}(\lambda^{\text{OPT}}) \times P_{stat} && t_{\max}(\lambda_g) = t_{\max}(\lambda^{\text{OPT}}) \\
&\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + b \times A_{stat}(\lambda^{\text{OPT}}) \times P_{stat} && p \times t_{\max} \leq A_{stat} \text{ for any given schedule} \\
&\leq \sum_i E_i(\lambda_g) + b \times A_{stat}(\lambda^{\text{OPT}}) \times P_{stat} && a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha \leq E_i(\lambda_g) \text{ from the definition of } E_i \\
&\leq \sum_i E_i(\lambda_g) + b \times E(\lambda^{\text{OPT}}) && A_{stat} \times P_{stat} \leq E \text{ for any given schedule} \\
&\leq \sum_i E_i(\lambda^{\text{OPT}}) + b \times E(\lambda^{\text{OPT}}) && \text{Equation (3)} \\
&\leq E(\lambda^{\text{OPT}}) + b \times E(\lambda^{\text{OPT}}) && \sum E_i \leq E \\
&\leq (b+1) \times E(\lambda^{\text{OPT}})
\end{aligned}$$

We finally reunite the two sides of the max:

$$\begin{aligned} E(\lambda_\bullet) &\leq \max\left(a \times E(\lambda^{\text{OPT}}), (b+1) \times E(\lambda^{\text{OPT}})\right) \\ &\leq \max\left(a, b+1\right) \times E(\lambda^{\text{OPT}}), \end{aligned}$$

which concludes the proof. □

In this case, the bound is 3 for both ListBased and ShelfBased algorithms.

6. Approximation ratios with continuous speeds

We propose a theoretical variation of the problem, where instead of choosing the speed in a finite set of speeds S, we can choose any speed in $\mathbb{R}_+^*$. We call this continuous problem MinE-Mold-Cont and MinE-Rig-Cont, depending on the nature of the tasks. The motivation for this variation is that we can get stronger approximation results through the introduction of continuous speeds. Through this, we hope to get a better understanding of what makes a good processor speed, thus allowing us to better choose the set of speeds S at the creation of a processor.

6.1. Rigid case

Similarly to the discrete case, we start by restricting to rigid tasks (MinE-Rig-Cont-SS problem).

Theorem 5. *In the rigid single-speed context with continuous speeds (MinE-Rig-Cont-SS) and assuming that there exists an algorithm $\mathcal{A}$ that yields a c-approximation of the optimal makespan, we can compute in polynomial time a schedule consuming at most $c^{1-\frac{1}{\alpha}}$ times the optimal energy.*

Proof. In the rigid case, we have $\sum_{i=1}^n a_{i,p_i,s_i} = \sum_{i=1}^n \frac{w_{i,p_i}}{s_i}$. The cumulative work $W = \sum_{i=1}^n w_{i,p_i}$ is independent of the schedule λ. With a single speed, we can hence write the energy as:

$$E(\lambda) = W \times s_\lambda^{\alpha-1} + A_{stat}(\lambda) \times P_{stat} \tag{4}$$

$$= W \times s_\lambda^{\alpha-1} + p \times C_{\max}(\lambda) \times P_{stat}. \tag{5}$$

The problem can be split as two decisions to take:

- The choice of speed s;
- The actual scheduling, i.e., the choice of the time at which we start each task.

To show that these decisions can be taken one after the other, we start with a preliminary lemma comparing the energy consumption of two schedules using the same speed.

Lemma 1. *Let λ_1, λ_2 be two single-speed schedules such that $s_{\lambda_1} = s_{\lambda_2}$. If $E(\lambda_1) \leq E(\lambda_2)$, then for any $\rho > 0$, $E(\rho\lambda_1) \leq E(\rho\lambda_2)$.*

Proof. Using Equation (2),

$$E(\lambda_2) - E(\lambda_1) = p \times P_{stat} \times (C_{\max}(\lambda_2) - C_{\max}(\lambda_1)).$$

Furthermore, $C_{\max}(\lambda_1) = \rho \times C_{\max}(\rho\lambda_1)$ and $C_{\max}(\lambda_2) = \rho \times C_{\max}(\rho\lambda_2)$. It follows that:

$$\begin{aligned} E(\lambda_2) - E(\lambda_1) &= p \times P_{stat} \times \rho \times (C_{\max}(\rho\lambda_2) - C_{\max}(\rho\lambda_1)) \\ &= \rho \times (E(\rho\lambda_2) - E(\rho\lambda_1)), \end{aligned}$$

hence proving the lemma. □

Lemma 1 shows that the actual scheduling can be expressed as a two-steps minimization problem: find a schedule λ_0 minimizing the makespan for a given speed. Next, find among $[\lambda_0]$ (using the notations defined in Section 5.1) a schedule (i.e., a speed) minimizing the energy consumption.

For a given schedule λ_0, we can compute the optimal speed s as the one that minimizes

$$f(s) = W \times s^{\alpha-1} + p \times \frac{K_{[\lambda_0]}}{s} \times P_{stat},$$

with $K_{[\lambda]} = s_\lambda \times C_{\max}(\lambda)$ as defined in Section 5.1. This optimal value of s is

$$s^{\text{OPT}}_{[\lambda_0]} \triangleq \sqrt[\alpha]{\frac{p \times K_{[\lambda_0]}}{(\alpha-1) \times W} \times P_{stat}}.$$

Then, we can write

$$\min_{\lambda \in [\lambda_0]} E(\lambda) = C \times \sqrt[\alpha]{K^{\alpha-1}_{[\lambda_0]} \times W},$$

where $C = \sqrt[\alpha]{(p \times P_{stat})^{\alpha-1}} \times (\sqrt[\alpha]{\alpha-1} + \sqrt[\alpha]{(\alpha-1)^{\alpha-1}})$ is a constant independent of λ_0.

Consequently, we have

$$\begin{aligned} E(\lambda^{\text{OPT}}) &= \min_{[\lambda_0] \sim \text{class}} \min_{\lambda \in [\lambda_0]} E(\lambda) \\ &= \min_{[\lambda_0] \sim \text{class}} C \times \sqrt[\alpha]{K^{\alpha-1}_{[\lambda_0]} \times W} \\ &= C \times \sqrt[\alpha]{K^{\alpha-1}_{[\lambda^*]} \times W}. \end{aligned}$$

Now, let $\mathcal{A}$ be an algorithm that yields a c-approximation for the makespan. For a given speed, the

quantity $K_{[\lambda_{\mathcal{A}}]}$ is proportional to the makespan, so this algorithm outputs a schedule $\lambda_{\mathcal{A}}$ such that

$$K_{[\lambda_{\mathcal{A}}]} \leq c \times K_{[\lambda^*]}.$$

By running this schedule with speed

$$s^{\text{OPT}}_{[\lambda_{\mathcal{A}}]} = \sqrt[\alpha]{\frac{p \times K_{[\lambda_0]}}{(\alpha-1) \times W} \times P_{stat}},$$

we have a schedule such that

$$\begin{aligned} E_{\lambda_{\mathcal{A}}} &= C \times \sqrt[\alpha]{K^{\alpha-1}_{[\lambda_{\mathcal{A}}]} \times W} \\ &\leq C \times \sqrt[\alpha]{\left(c \times K_{[\lambda^*]}\right)^{\alpha-1} \times W} \\ &\leq c^{\frac{\alpha-1}{\alpha}} \times E(\lambda^{\text{OPT}}). \end{aligned}$$

This proves the theorem: an algorithm $\mathcal{A}$, that yields a c-approximation for the makespan and that returns a schedule $\lambda_{\mathcal{A}}$ will yield a $c^{\frac{\alpha-1}{\alpha}}$-approximation for the energy. □

6.2. Moldable case

The overall design of the algorithm for MinE-Mold-Cont-SS is similar to the one of the discrete case, see Algorithm 3.

Algorithm 3: Algorithm for MinE-Mold-Cont-SS

for $(T_{i'}, p') \in \{T_1, \ldots, T_n\} \times \{1, \ldots, p\}$ **do**
  $t_{\max} \leftarrow t_{i',p'}$;
  for $T_i \in \{T_1, \ldots, T_n\}$ **do**
    $p_i \leftarrow \arg\min_{1 \leq j \leq p} j \times t_{i,j}$ such that $t_{i,p_i} \leq t_{\max}$ if it exists;
    $p_i = 0$ otherwise;
  if *all* $p_i \neq 0$ **then**
    Apply a guaranteed algorithm $\mathcal{A}$ on the rigid instance $\{(T_1, p_1), \ldots, (T_n, p_n)\}$ at speed of 1, to get a schedule $\lambda_\bullet^{(T_{i'}, p')}$;
    Select the speed $s_\bullet \stackrel{\Delta}{=} \sqrt[\alpha]{\frac{p \times K_{[\lambda_\bullet]}}{(\alpha-1) \times W} \times P_{stat}}$ for this schedule ;
return the schedule $\lambda_\bullet$ with minimum energy among all computed $\lambda_\bullet^{(T_{i'}, p')}$;

We can state the main result of this section.

Theorem 6. *We assume that there exists a polynomial-time algorithm $\mathcal{A}$ for MinE-Rig-Cont-SS that returns a schedule $\lambda_{\mathcal{A}}$ at a speed of one such that $C_{\max}(\lambda_{\mathcal{A}}) \leq a \times \frac{W(\lambda_{\mathcal{A}})}{p} + b \times t_{\max}(\lambda_{\mathcal{A}})$ (resp. $C_{\max}(\lambda_{\mathcal{A}}) \leq \max\left(a \times \frac{W(\lambda_{\mathcal{A}})}{p}, b \times t_{\max}(\lambda_{\mathcal{A}})\right)$). In the moldable single-speed context with continuous speeds (MinE-Mold-*

Cont-SS), one can compute in polynomial time a schedule $\lambda_\bullet$ that consumes at most $(a+b)^{1-\frac{1}{\alpha}}$ (resp. $\max(a,b)^{1-\frac{1}{\alpha}}$) times the optimal energy, when running at speed $s_\bullet \triangleq \sqrt[\alpha]{\frac{p \times K_{[\lambda_\mathcal{A}]}}{(\alpha-1)\times W} \times P_{stat}}$.

Proof. We adapt the proof of Theorem 3 to continuous speeds for the case $C_{\max}(\lambda_\mathcal{A}) \leq a \times \frac{W(\lambda_\mathcal{A})}{p} + b \times t_{\max}(\lambda_\mathcal{A})$. The other case (maximum of the two terms instead of sum) is similar.

For any task i and any number p_i of processors, we denote by λ_{i,p_i} the schedule returned by $\mathcal{A}$ on the following rigid instance: for all i', $p_{i'}$ is the integer in $\{1,\ldots,p\}$ minimizing $w_{i',p_{i'}}$ under the constraint $t_{i',p_{i'}} \leq t_{i,p_i}$. There are at most np different such schedules and each one can be computed in polynomial time. For each schedule, the selected speed is the one that minimizes the energy consumption. Let $\lambda_\bullet$ be a schedule of $(\lambda_{i,p_i})_{i,p_i}$ (where each task is running at speed $s_{i,p_i} = \sqrt[\alpha]{\frac{K_{[\lambda_{i,p_i}]}}{(\alpha-1)\times W} \times P_{stat}}$) with minimum energy consumption.

Let λ^{OPT} be a schedule minimizing the energy (for MinE-Mold-Cont-SS). Let i^{OPT} and $p_{i^{\text{OPT}}}$ denote the task and the number of processors, such that $t_{\max}(\lambda^{\text{OPT}}) = t_{i^{\text{OPT}},p_{i^{\text{OPT}}}}$ in the schedule λ^{OPT} that minimizes the energy consumption. By construction of the λ_{i,p_i}, one has $W(\lambda_g) \leq W(\lambda^{\text{OPT}})$ where $\lambda_g = \lambda_{i^{\text{OPT}},p_{i^{\text{OPT}}}}$. As in the rigid context (proof of Theorem 5), we can express the energy for the respective schedules as $E(\lambda_g = C \sqrt[\alpha]{K_{[\lambda_g]}^{\alpha-1} \times W(\lambda_g)}$ and $E(\lambda^{\text{OPT}}) = C \sqrt[\alpha]{K_{[\lambda^{\text{OPT}}]}^{\alpha-1} \times W(\lambda^{\text{OPT}})}$ where $C = \sqrt[\alpha]{(p \times P_{stat})^{\alpha-1}} \times (\sqrt[\alpha]{\alpha-1} + \sqrt[\alpha]{(\alpha-1)^{\alpha-1}})$.

Consequently $\frac{E(\lambda_\bullet)}{E(\lambda^{\text{OPT}})} \leq \frac{E(\lambda_g)}{E(\lambda^{\text{OPT}})} = \sqrt[\alpha]{\frac{K_{[\lambda_g]}^{\alpha-1}}{K_{[\lambda^{\text{OPT}}]}^{\alpha-1}}}$. Now, we have

$$\begin{aligned} K_{[\lambda_g]} &\leq a \times \frac{W(\lambda_g)}{p} + b \times t_{\max}(\lambda_g) \\ &\leq a \times \frac{W(\lambda^{\text{OPT}})}{p} + b \times t_{\max}(\lambda^{\text{OPT}}) \\ &\leq (a+b) \times K_{[\lambda^{\text{OPT}}]} \end{aligned}$$

From that, we finally get

$$\frac{E(\lambda_\bullet)}{E(\lambda^{\text{OPT}})} \leq \sqrt[\alpha]{(a+b)^{\alpha-1}},$$

which concludes the proof. □

Recall that ListBased-SS is an algorithm with a maximum and $a = 2$ and $b = 2$ [13], hence it is a $2^{1-\frac{1}{\alpha}}$-approximation algorithm. ShelfBased-SS, an algorithm with a sum and $a = 2$ and $b = 1$ [32, LTF], is thus a $3^{1-\frac{1}{\alpha}}$-approximation algorithm. For $\alpha = 3$, these approximation ratios become respectively $\sqrt[3]{4} \approx 1.59$ and $\sqrt[3]{9} \approx 2.08$.

7. Optimizing for a single shelf

We propose to further optimize co-schedules by designing a polynomial-time algorithm for the MinE-OneShelf problem, i.e., to optimize the execution of a single shelf, both with discrete and continuous speeds. Formally, given a set of n tasks and p processors, the goal is to find an assignment $((p_i),(s_i))_{1\le i\le n}$ that minimizes $E=\sum_{i=1}^{n}\left(p_i\times t_{i,dyn}\times s_i^{\alpha}+p_i\times t_{i,stat}\times P_{stat}\right)$, where

- $t_{i,dyn}=t_{i,p_i,s_i}=\frac{t_{i,p_i}}{s_i}$, and
- $t_{i,stat}=\max_{1\le i\le n} t_{i,dyn}$ (*simultaneous* model).

7.1. Preliminaries

Since the static energy spent depends on the total length of the shelf (i.e., $\max_{1\le i\le n} t_{i,p_i,s_i}$), the algorithm proceeds by fixing the shelf length to $C_{\max}$, and aims at finding the optimal number of processors and speed for each task, such that the time bound $C_{\max}$ is respected and the total energy consumption is minimized.

Hence, for a single processor, given an amount of work w to complete and a length of shelf of $C_{\max}$, we consider the function $OptS(w,C_{\max})$ that returns the optimal speed such that $\frac{w}{s}\le C_{\max}$ and the energy consumption $w\times s^{\alpha-1}+C_{\max}\times P_{stat}$ is minimized. Since the energy consumption is an increasing function of s for $s\ge 0$, the optimal speed is the smallest speed such that the shelf length is not exceeded. Therefore, $OptS(w,C_{\max})=\max(s_{\min},\frac{w}{C_{\max}})$ in the continuous case, and $OptS(w,C_{\max})=\min\left\{s\in S\mid s\ge\frac{w}{C_{\max}}\right\}$ in the discrete case. In the case no such speed exists (this may happen with discrete speeds), the function returns None. Note that this function can be computed in $\Theta(1)$ in the continuous case, and in $\Theta(\log(|S|))$ in the discrete case by doing a binary search within values of S.

7.2. Optimal algorithm for MinE-OneShelf (discrete speeds)

We first focus on the discrete case, i.e., S is the set of possible speeds. The idea is to try every possible duration of the shelf: all possible durations are recorded in the set $\mathcal{T}$, and then for a given duration $C_{\max}\in\mathcal{T}$, we compute the solution for each set of tasks $T_1,\ldots,T_i$, $i\in[\![1,n]\!]$ and each number of processors $q\in[\![1,p]\!]$.

Let $e_{i,q}$ be the minimum energy consumed by task T_i on q processors, while not exceeding time $C_{\max}$. It is computed by using the function $OptS(t_{i,q},C_{\max})$, since $t_{i,q}$ is the amount of work on one processor if task T_i is executed on q processors.

We then proceed with a dynamic programming algorithm, to compute $E_{i,q}$, the minimum energy consumption for the first i tasks, when using a total of q processors. The goal is to compute $E_{n,p}$ (using all tasks and all processors). $E_{i,q}$ is recursively defined for $1\le i\le n$ and $1\le q\le p$ as:

$$E_{i,q}=\min_{1\le k\le q-i+1} E_{i-1,q-k}+e_{i,k},$$

with $E_{0,q} = 0$. If there are no tasks left, the energy consumption is null; otherwise we try all possible numbers of processors k for task i, while keeping at least one processor for each of the remaining tasks.

We then take the best possible solution amongst the different possible durations in the set $\mathcal{T}$, and Algorithm 4 provides the corresponding pseudo-code of this dynamic programming algorithm.

Algorithm 4: Optimal algorithm for MinE-OneShelf (discrete model)

```
1  T ← ∅ ;
2  for i ← 1 to n do
3      for q ← 1 to p do
4          for s ∈ S do
5              T ← T ∪ { t_{i,q} / s } ;
6  res ← ∞ ;
7  for C_max ∈ T do
8      for i ← 1 to n do
9          for q ← 1 to p do
10             s_{i,q} ← OptS(t_{i,q}, C_max) ;
11             if s_{i,q} = None then
12                 e_{i,q} ← ∞ ;
13             else
14                 e_{i,q} ← q t_{i,q} s_{i,q}^{α−1} + q C_max P_stat ;
15     for q ← 0 to p do
16         E_{0,q} ← 0 ;
17     for i ← 1 to n do
18         for q ← i to p do
19             for k ← 1 to q − i + 1 do
20                 if E_{i−1,q−i} + e_{i,k} < E_{i,q} then
21                     E_{i,q} ← E_{i−1,q−k} + e_{i,k} ;
22     if E_{n,p} < res then
23         res ← E_{n,p}
24 return res ;
```

Theorem 7. *MinE-OneShelf can be solved optimally in polynomial time (discrete model).*

Proof. Let us prove by induction over $i \in [\![0, n]\!]$ that for all $q \in [\![1, p]\!]$, $E_{i,q}$ is the minimum energy consumed to process the i first tasks with q processors.

Base case. For all $q \in [\![0, p]\!]$, the energy consumed to handle no task on q processors is 0, meaning that the $E_{0,q}$ values for $q \in [\![0, p]\!]$ are correct.

Inductive step. Let $i \in [\![0, n-1]\!]$, and we assume that $\forall q \in [\![1, p]\!]$, $E_{i,q}$ is correct. The expression of $E_{i+1,q}$ is $\min_{1 \leq k \leq q-i} E_{i,q-k} + e_{i+1,k}$. If task T_{i+1} is given k processors, then the i first tasks will be handled by $q - k$ processors. As the task $i + 1$ must be given a number of processors $k \in [\![1, q - i]\!]$, the expression gives the correct value for $E_{i+1,q}$.

It means that $E_{n,p}$ is correct, and therefore that the algorithm is also correct.

The number of total durations is at most $np|S|$, because we must choose a task, the number of processors allocated for this task, and its speed. The complexity of the algorithm is thus $O(n^2p^3|S|)$. □

7.3. Optimal algorithm for MinE-OneShelf-Cont (continuous speeds)

We now discuss the case of continuous speeds, hence $S = \mathbb{R}_+^*$. Similarly to the discrete case, the idea is to fix the shelf duration and to solve the problem knowing that the processors will be powered for the duration $C_{\max}$. Because of continuous speeds, we cannot anymore explore all possible times $C_{\max}$, so we fix the duration to $C_{\max} = 1$, and then prove that the optimal assignment is in fact the same for any $C_{\max}$, and the minimum energy is a function of $C_{\max}$. We finally take the value of $C_{\max}$ that minimizes the energy consumption, see Algorithm 5 and Theorem 8 for the proof of optimality.

Algorithm 5: Optimal algorithm for MinE-OneShelf-Cont

```
for i ← 1 to n do
    for q ← 1 to p do
        s_{i,q} ← OptS(t_{i,q}, 1) ;
        if s_{i,q} = None then
            e_{i,q} ← ∞ ;
        else
            e_{i,q} ← q t_{i,q} s_{i,q}^{α−1} + q P_stat ;
for q ← 0 to p do
    E_{0,q} ← 0 ;
for i ← 1 to n do
    for q ← i to p do
        for k ← 1 to q − i + 1 do
            if E_{i−1,q−k} + e_{i,k} < E_{i,q} then
                E_{i,q} ← E_{i−1,q−k} + e_{i,k} ;
C_max ← ∛((α − 1)(E_{n,p} / (p P_stat) − 1));
return (E_{n,p} − p P_stat) / C_max^{α−1} + p C_max P_stat ;
```

Lemma 2. *Let $E_{dyn}(C_{\max})$ be the minimum possible dynamic energy consumption, with the condition that each task must end before $C_{\max}$. Then, for any $C_{\max} \in \mathbb{R}_+^*$, we have:*

$$E_{dyn}(C_{\max}) = \frac{E_{dyn}(1)}{C_{\max}^{\alpha-1}}.$$

Proof. Recall that the dynamic energy consumption of an assignment $((p_i),(s_i))_{1\le i\le n}$ is $\sum_{i=1}^{n} p_i \times t_{i,p_i} \times s_i^{\alpha-1}$.

Let $t \in \mathbb{R}_+^*$. If $((p_i),(s_i))_{1\le i\le n}$ is an optimal assignment for the case $C_{\max} = 1$, with a total dynamic energy consumption of $E_{dyn}(1)$, then we can consider the same assignment but with all speeds divided by t, to ensure that all tasks meet the deadline $C_{\max} = t$: $((p_i),(\frac{s_i}{t}))_{1\le i\le n}$. The corresponding dynamic energy consumption is then $\frac{E_{dyn}(1)}{t^{\alpha-1}}$, and hence the optimal solution $E_{dyn}(t)$ is such that $E_{dyn}(t) \le \frac{E_{dyn}(1)}{t^{\alpha-1}}$.

Problem	Base ratio ($\mathcal{A}$)	Achieved bound	Result
MinE-Mold-Indep	OPT	OPT	Sec. 4.1
MinE-Rig-SS	c	c	Th. 2 (Sec. 5.1.1)
MinE-Mold-SS	$a\frac{W}{p} + bt_{\max}$	$a + b$	Th. 3 (Sec. 5.1.2)
	$\max(a\frac{W}{p}, bt_{\max})$	$\max(a, b)$	
MinE-Mold	$a\frac{W}{p} + bt_{\max}$	$a + b$	Th. 4 (Sec. 5.2)
	$\max(a\frac{W}{p}, bt_{\max})$	$\max(a, b+1)$	
MinE-Rig-Cont-SS	c	$c^{1-\frac{1}{\alpha}}$	Th. 5 (Sec. 6.1)
MinE-Mold-Cont-SS	$a\frac{W}{p} + bt_{\max}$	$(a+b)^{1-\frac{1}{\alpha}}$	Th. 6 (Sec. 6.2)
	$\max(a\frac{W}{p}, bt_{\max})$	$\max(a, b)^{1-\frac{1}{\alpha}}$	
MinE-OneShelf		Optimal	Th. 7 (Sec. 7.2)
MinE-OneShelf-Cont		Optimal	Th. 8 (Sec. 7.3)

Table 3: Summary of theoretical results from Section 4to Section 7. The base ratio is the approximation ratio on the makespan of the base algorithm $\mathcal{A}$.

Conversely, if we have an assignment for the problem with $C_{\max} = t$, with a dynamic energy consumption of $E_{dyn}(t)$, then we take the same assignment but with all speeds multiplied by t to obtain a valid solution to the problem with $C_{\max} = 1$, hence leading to $E_{dyn}(t) \geq \frac{E_{dyn}(1)}{t^{\alpha-1}}$.

This concludes the proof of the lemma since $E_{dyn}(t) = \frac{E_{dyn}(1)}{t^{\alpha-1}}$ for any $t \in \mathbb{R}_+^*$. □

Theorem 8. *MinE-OneShelf-Cont can be solved optimally in polynomial time.*

Proof. The proof for a fixed $C_{\max}$ is the same as in the proof of Theorem 7, since we use the same dynamic programming algorithm. Then, $E_{n,p} = E_{dyn}(1) + p \times P_{stat}$. From Lemma 2, the optimal energy for a given $C_{\max}$ is therefore $E(C_{\max}) = E_{dyn}(C_{\max}) + E_{stat}(C_{\max}) = \frac{E_{n,p} - p \times P_{stat}}{C_{\max}^{\alpha-1}} + p \times C_{\max} \times P_{stat}$, which is a convex function of $C_{\max}$ that reaches its minimum for $C_{\max} = \sqrt[\alpha-2]{(\alpha-1)(\frac{E_{n,p}}{p \times P_{stat}} - 1)}$. The complexity of the algorithm is $O(np^2)$. □

The next section empirically assesses the theoretical results summarized in Table 3.

8. Empirical Study

We first describe the experimental setup in Section 8.1. Then, we explain how instances are generated in Section 8.2. The different heuristics are compared and analyzed in Section 8.3, Also, we further study the impact of P_{stat} in Section 8.4, and the impact of having a set discrete speeds instead of the continuous model in Section 8.5.

8.1. Experimental setup

All the algorithms rely on the global mechanism presented in Algorithm 1 (Section 5.1.2 [32, GF]) with a single speed (denoted with the suffix SS) and Algorithm 2 with multiple speeds (without any suffixes). The

Algorithm 6: Algorithm ListBased for rigid tasks $\{(T_1, p_1, s_1), \ldots, (T_n, p_n, s_n)\}$

```
λ ← Empty schedule;
for j ∈ {1, ... p} do
    C_j ← 0;
𝒯 ← {T_1, ..., T_n};
𝒫 ← ∅;
C_current ← 0;
while 𝒯 ≠ ∅ do
    if ∃T_i ∈ 𝒯 s.t. p_i ≤ |𝒫| then
        i ← min_{T_i ∈ 𝒯} i s.t. p_i ≤ |𝒫|;
        λ ← λ ∪ {T_i starting at time C_current on p_i processors from 𝒫};
        for k ∈ {1, p_i} do
            Let j ∈ 𝒫;
            C_j ← C_current + t_{i,p_i,s_i};
            𝒫 ← 𝒫\{j};
        𝒯 ← 𝒯\{T_i};
    else
        j ← arg min_{j ∈ {1,...,p}\𝒫} C_j;
        C_current ← C_j;
        while C_current = min_{j ∈ {1,...,p}\𝒫} C_j do
            j ← min_{j ∈ {1,...,p}\𝒫} C_j;
            𝒫 ← 𝒫 ∪ {j};
return the schedule λ;
```

Algorithm 7: Algorithm ShelfBased for rigid tasks $\{(T_1, p_1, s_1), \ldots, (T_n, p_n, s_n)\}$

```
Sort the tasks by non increasing execution time so that T_1 has the longest execution time and T_n
  the shortest;
λ ← Empty schedule;
C_current ← 0;
C_next ← 0;
𝒯 ← {T_1, ..., T_n};
𝒫 ← ∅;
for T_i ∈ 𝒯 by increasing i do
    if p_i > |𝒫| then
        C_current ← C_next;
        C_next ← C_current + t_{i,p_i,s_i};
        𝒫 ← {1, ..., p};
    λ ← λ ∪ {T_i starting at time C_current on p_i processors from 𝒫};
    for k ∈ {1, p_i} do
        𝒫 ← 𝒫\{j};
return the schedule λ;
```

core idea is that first we transform a moldable instance into a rigid instance by fixing the number of processors for each task. It is then combined with the strategies presented in Section 2: ListBased and ShelfBased. The two algorithms ListBased and ShelfBased are detailed as Algorithms 6 and 7. Moreover, we also implemented two optimization algorithms that can only be applied to an output of ShelfBased:

- OptiShelf, which optimizes each shelf once each task has been allocated to a shelf using the algorithm from Section 7 (this may change the number of processors used for each task). This optimization keeps the shelf structure, which can be an advantage for instances where this structure is a constraint the final schedule is subject to;
- De-shelf, which takes a ShelfBased solution and starts each task as soon as possible by removing the shelf constraint while keeping the allocations and the order in which the tasks are started.

For both of these optimizations, the energy consumption cannot be worse after the optimization than before. It is technically possible to combine the two optimizations (running OptiShelf and then De-shelf). We tried it for the sake of completeness, however this did not provide any interesting results as OptiShelf's optimization is heavily based on the shelf structure, while De-shelf removes this shelf structure. Overall, this represents a total of eight heuristics (two list-based, two non-optimized shelf-based, and four optimized versions of shelf-based).

To compare the different heuristics, we implemented them in C++17 compiled with `gcc` 9.3.0 with optimization option `-O3`. We rely on Python 3.8.5 to generate the instances and to analyze the results. The code of these experiments can be found on Figshare[1].

8.2. Instance generation

The characteristics of the processors were extracted from a realistic platform [24, 7]:

Processor	p	P_{stat}	α	S
Intel Xscale	32	$\frac{6}{155} \approx 3.9 \times 10^{-2}$	3	$\{0.15, 0.4, 0.6, 0.8, 1\}$
Transmeta Crusoe	32	$\frac{44}{57560} \approx 7.6 \times 10^{-4}$	3	$\{0.45, 0.6, 0.8, 0.9, 1\}$

The number of tasks varies from 20 to 1000, with a step every 20 tasks. The workload was generated with the two following task profiles (half from each type):

- Amdahl's law [1, 30]: $w_{i,p_i} = w_{i,1} \times \beta + \frac{w_{i,1} \times (1-\beta)}{p_i}$;
- Power law [26, 16, 30]: $w_{i,p_i} = \frac{w_{i,1}}{p_i^{\beta}}$.

In both cases, $w_{i,1}$ and β are drawn from a uniform distribution $U(0, 1)$.

[1] https://doi.org/10.6084/m9.figshare.14854395

8.3. Results

In order to evaluate the performance of the various heuristics and show whether they return results close to the optimal, we compare the results with a lower bound that consists in an optimal execution in the MinE-Mold-Indep case (Section 4.1). In that case, the static energy is paid only while a task is executed, and any solution to MinE-Mold will consume at least as much energy as this lower bound. For convenience, the default version of a heuristic is the multiple-speed variant, and we refer to the single-speed variant with the SS suffix.

Figures 1 and 2 present an overview of the results for all heuristics with $n = 500$ tasks, respectively on the Intel Xscale platform and on the Transmeta Crusoe platform. We report both the ratio between the

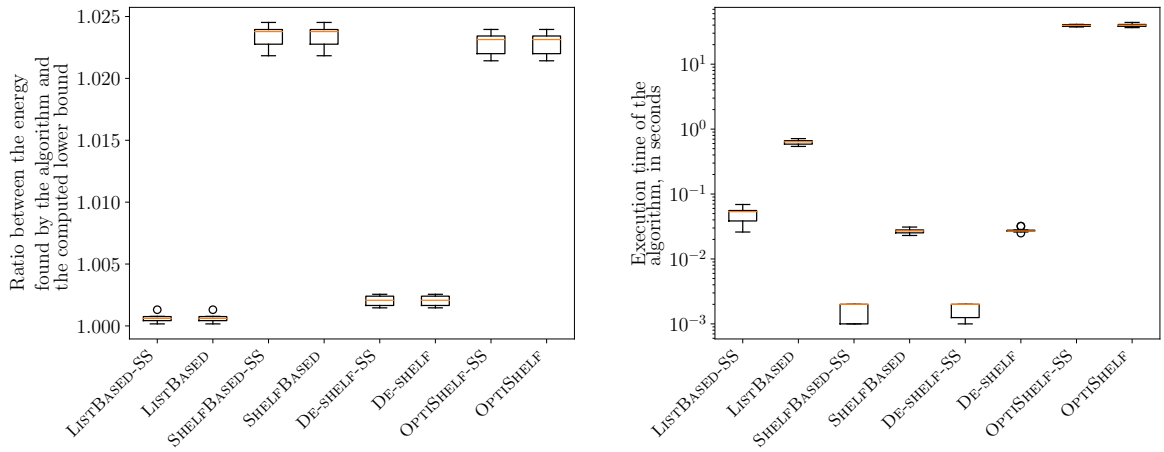

Figure 1: Output energy consumption and execution time to compute the solution for all eight heuristics with $n = 500$ mixed power and Amdahl's tasks and on the Intel Xscale platform ($p = 32$ processors with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$). Each box aggregates 10 measurements.

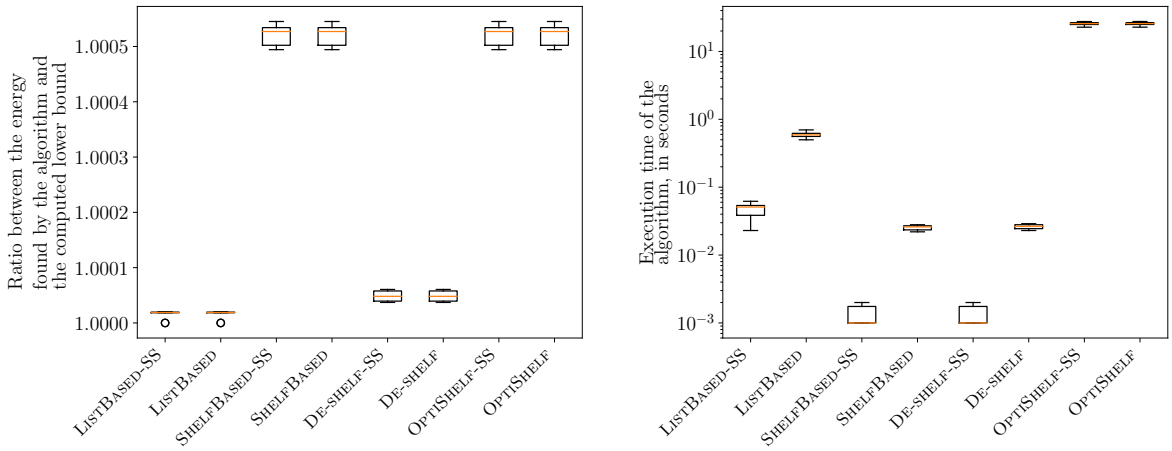

Figure 2: Output energy consumption and execution time to compute the solution for all eight heuristics with $n = 500$ mixed power and Amdahl's tasks and on the Transmeta Crusoe ($p = 32$ processors with $P_{stat} = \frac{44}{57560}$, $\alpha = 3$, $S = \{0.45, 0.6, 0.8, 0.9, 1\}$). Each box aggregates 10 measurements.

energy consumption of each heuristic with the lower bound (the lower the better), and also the execution time of the C++ implementation of the heuristics. A first remark is that single-speed and multiple-speed variants give very similar results. Indeed, in practice, we could confirm that the multiple-speed heuristics give the same speed to most tasks.

Figures 3 and 4 present a similar overview with a larger amount of tasks: $n = 5000$ tasks per instance. The difference in energy consumption between ListBased and De-shelf becomes smaller as n increases, while the execution time of the algorithm ListBased becomes much larger due to a higher order of growth.

Figures 5 and 6 present the scaling with n of all heuristics respectively on the Intel Xscale platform and the Transmeta Crusoe platform. With a large number of tasks, the ratio with the lower bound becomes very

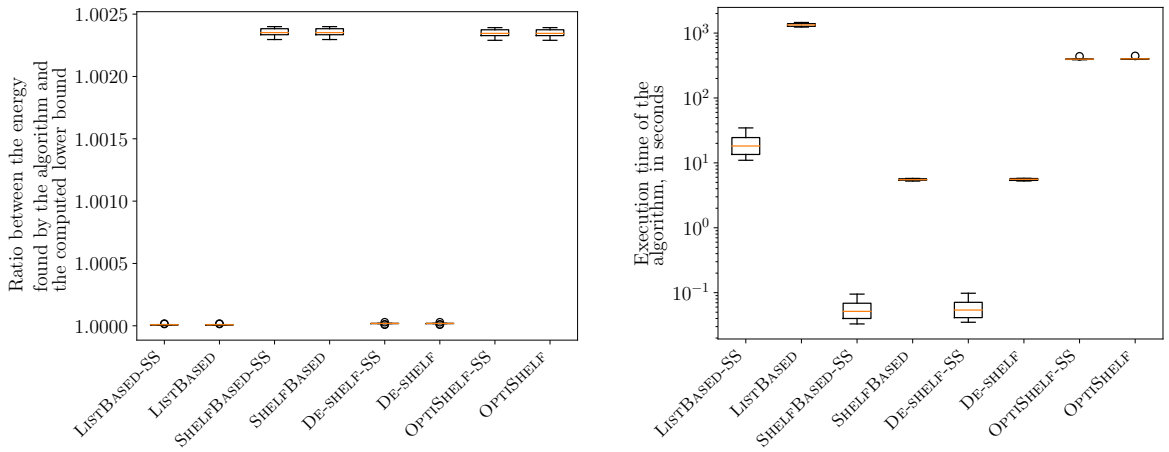

Figure 3: Output energy consumption and execution time to compute the solution for all eight heuristics with $n = 5000$ mixed power and Amdahl's tasks and on the Intel Xscale platform ($p = 32$ processors with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$). Each box aggregates 10 measurements.

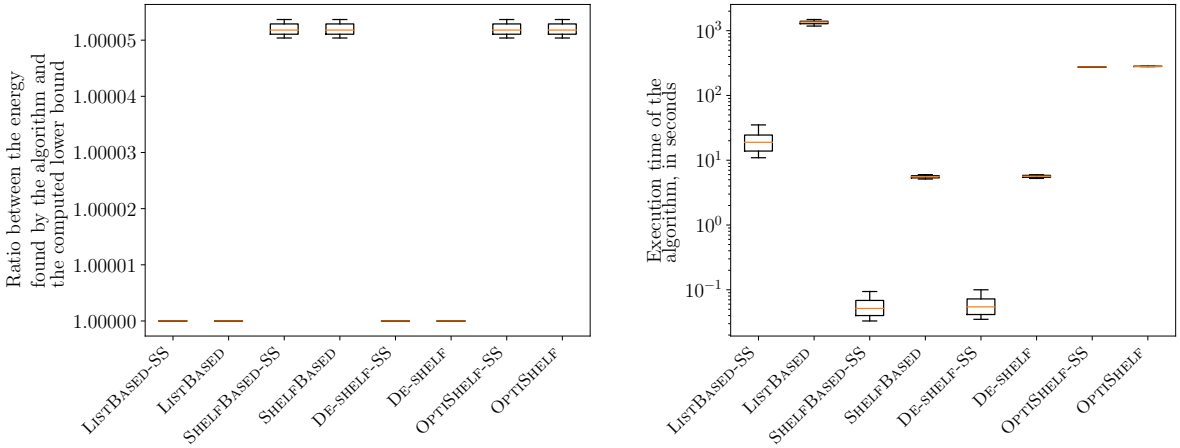

Figure 4: Output energy consumption and execution time to compute the solution for all eight heuristics with $n = 5000$ mixed power and Amdahl's tasks and on the Transmeta Crusoe ($p = 32$ processors with $P_{stat} = \frac{44}{57560}$, $\alpha = 3$, $S = \{0.45, 0.6, 0.8, 0.9, 1\}$). Each box aggregates 10 measurements.

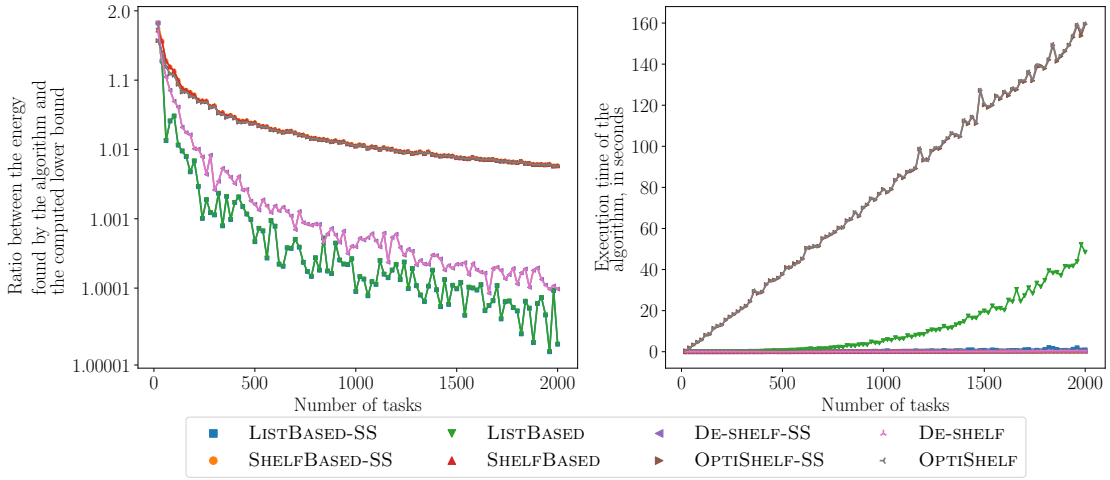

Figure 5: Output energy and execution time to compute the solution for all algorithms for instances with mixed power and Amdahl's tasks and and on the Intel Xscale platform ($p = 32$ processors with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$). The output energy is given with a $x \mapsto \log_{10}(x-1)$-scale.

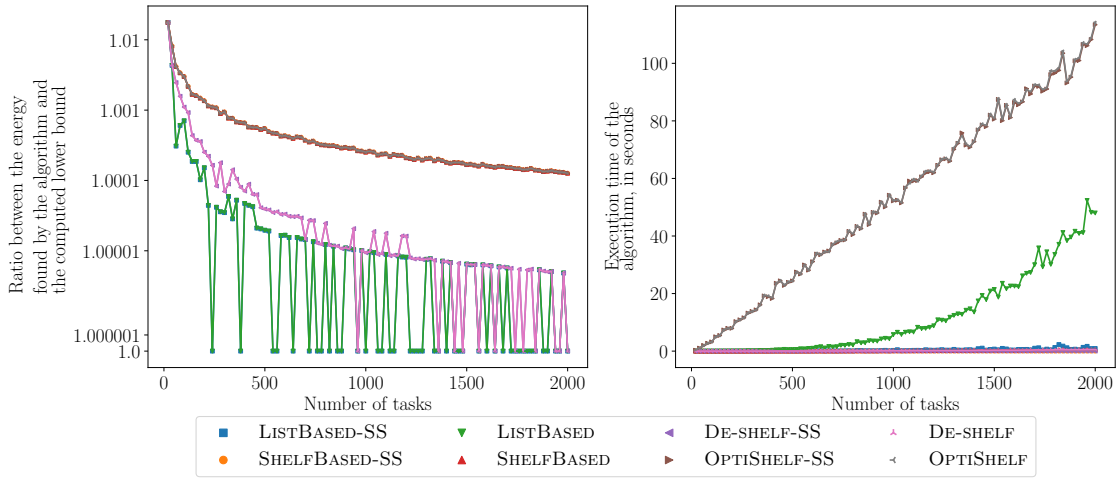

Figure 6: Output energy and execution time to compute the solution for all algorithms for instances with mixed power and Amdahl's tasks and and on the Transmeta Crusoe platform ($p = 32$ processors with $P_{stat} = \frac{44}{57560}$, $\alpha = 3$, $S = \{0.45, 0.6, 0.8, 0.9, 1\}$). The output energy is given with a $x \mapsto \log_{10}(x-1)$-scale.

close to 1 for all heuristics, at the price of an increasing execution time.

In terms of energy consumption, the best performing algorithms are ListBased and ListBased-SS (the lowest lines on the left charts). Then De-shelf and De-shelf-SS have a performance that is close to the ones of our best algorithms. Finally ShelfBased, ShelfBased-SS, OptiShelf and OptiShelf-SS have the worst performance among our algorithms (the top lines on the left charts).

In terms of execution time of the algorithms, most of our algorithms give instantaneous results. The exceptions are ListBased, that has a superlinear complexity with respect to the number of tasks, and both OptiShelf and OptiShelf-SS, that have a linear complexity with respect to the number of tasks but with a high constant factor.

As there are many algorithms and plots are overlapping, we then compare them in a more refined study, presenting more precisely how each algorithm behaves, along with the advantages and drawbacks of these algorithms.

Among the base algorithms (ListBased-SS, ListBased, ShelfBased-SS and ShelfBased), we focus on two baseline algorithms:

- the base algorithm with the best output energy: ListBased;
- the base algorithm with the best execution time: ShelfBased-SS.

If we compare these two algorithms, ListBased and ShelfBased-SS, we can see that ListBased provides schedules with a lower energy consumption than ShelfBased-SS, but at the cost of a much larger execution time for the algorithm. When the number of tasks n increases, the difference in terms of execution time increases, while the difference of energy consumption decreases. Note that ListBased could be implemented in a faster way with a segment tree to compute which task can be started, making this operation $O(\log p)$ instead of $O(n)$. However, this complex data structure would probably not be included in most implementations.

However, we can use the solution delivered by ShelfBased (with a single speed for all tasks), and pass this solution through two possible optimizations: OptiShelf or De-shelf. By comparing the results of the two approaches, we can see that both optimizations increase the quality of the solution, but OptiShelf does it at the cost of a very large increase in the execution time. However, the overhead of De-shelf is small, which leads to solutions of better quality at a small cost.

Finally, we compare ListBased (with multiple speeds) to the optimized ShelfBased-SS (with a single speed) with De-shelf, which we found to be the best optimization for ShelfBased. As we can see on Figures 5 and 6, for very small instances, ListBased still performs better than ShelfBased with De-shelf. However, when n grows larger, ShelfBased with De-shelf quickly performs as well as ListBased, but with a lower time complexity.

Overall, by comparing the results we get with the processors Intel Xscale and Transmeta Crusoe, we

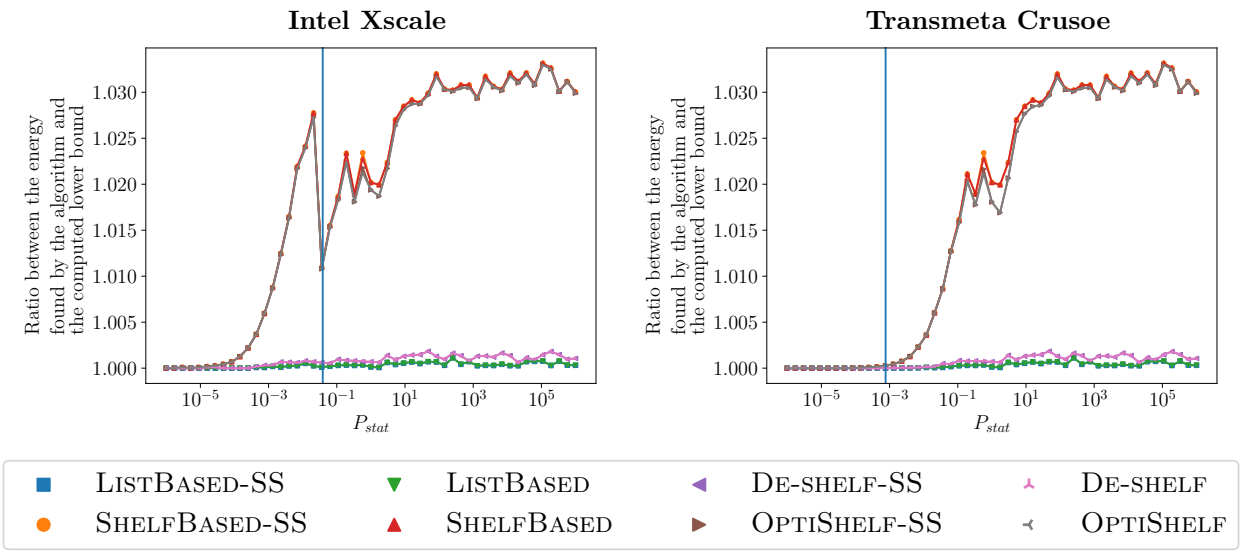

Figure 7: Output energy consumption to compute the solution for a variety of algorithms for instances with mixed power and Amdahl's tasks and $p = 32$ processors (on the left: with Intel Xscale $\alpha = 3$, and $S = \{0.15, 0.4, 0.6, 0.8, 1\}$; on the right: with Transmeta Crusoe $\alpha = 3$, and $S = \{0.45, 0.6, 0.8, 0.9, 1\}$). The vertical plain line corresponds to the actual P_{stat} of the studied processor.

see that, for all of the algorithms we provide, the schedules given for the Transmeta Crusoe are closer to the lower bound. First, this can be explained by the fact that the Transmeta Crusoe is a processor with a very low relative static power (around 7.6×10^{-4}) while the relative static power of the Intel Xscale is higher (around 3.9×10^{-2}). It means that there is less need to optimize the makespan, thus simplifying the problem. We explore the impact of P_{stat} in Section 8.4. Another explanation can be related to the difference of available speeds between the processors. We explore the impact of available speeds in the Section 8.5 through an empirical study of the continuous case.

8.4. Impact of P_{stat}

Figure 7 compares all the proposed algorithms when varying P_{stat}. We can observe that for small values of P_{stat}, even the algorithms that were not so efficient before provide good results. That is because, in this case, the idle time of the processors does not have a high cost in terms of energy consumption. Thus, having a small total amount of work W is more important than having a small makespan $C_{\max}$. Since all the algorithms try at some point to minimize W in the same way, they end up by all providing similar results.

However, when P_{stat} increases, the importance of minimizing the makespan $C_{\max}$ increases. The different algorithms provide different performance in terms of makespan, which explains the difference in performance: ListBased and optimized ShelfBased algorithms provide much better solutions than simple ShelfBased algorithms.

8.5. Comparison with the continuous relaxation

Finally, we conduct experiments with the relaxed continuous version of the problem, MinE-Mold-Cont, where the speed of the processors can be any positive number. The previous lower bound does not apply for this relaxed problem because there is no constraint on the minimum speed. The ratio between the energy consumption achieved by the algorithm and the lower bound can thus be lower than 1. Note that we focus here on single speed variants of the algorithms, where a single continuous speed will hence be chosen.

Figure 8 compares all of the discrete speed algorithms we propose to the algorithm ListBased-Cont we use for the relaxed problem. This algorithm gives a solution with continuous speeds that consumes 15% less energy than the best result we can get with the speeds available for the Intel Xscale. It means that with a better choice of speeds when designing the processor, we can expect a 15% decrease in energy consumption with our algorithms. For the Transmeta Crusoe processor, the energy gap is even bigger: with a better choice of speeds, we can hope to gain more than 80% of energy.

Figure 9 compares the discrete-speed algorithm with the best results (ListBased-SS) to the result we can get with continuous speeds, for different values of P_{stat}. Intuitively, all approaches are close to 1 when they rely on speeds that are close to the discrete speeds of the studied processor: in these cases the speeds used by the continuous speed algorithm are already available in the discrete speed model. When we get further away from these cases, we see that allowing continuous speeds would allow for a much better performance. It means that the design of the static power P_{stat} and the set of available speeds S must be done concordantly. This design can be helped by theoretical results, such as the ones we provide in this paper, along with simulations such as the ones we provide in this section.

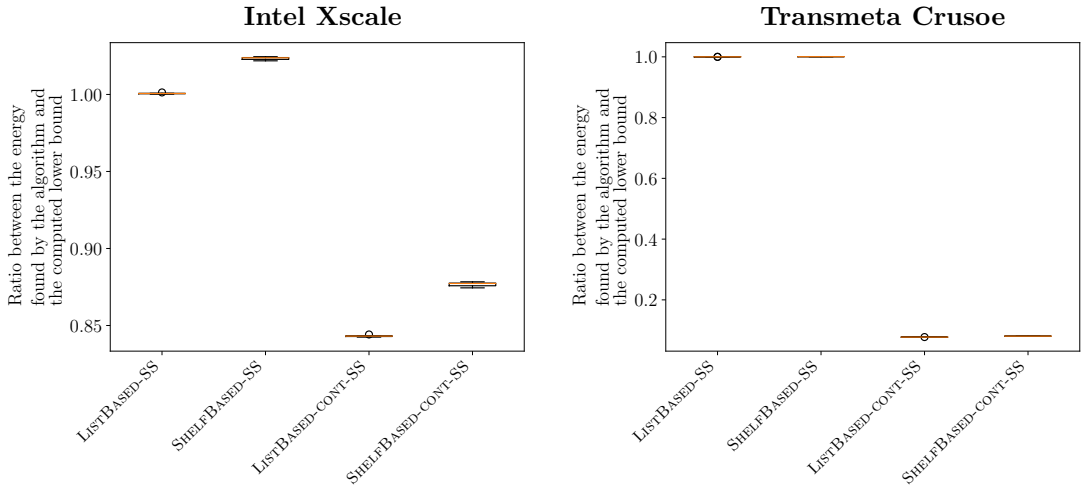

Figure 8: Output energy consumption and execution time to compute the solution for four heuristics with $n = 500$ mixed power and Amdahl's tasks and $p = 32$ processors (on the left: with Intel Xscale $P_{stat} = \frac{6}{155}$, $\alpha = 3$, and $S = \{0.15, 0.4, 0.6, 0.8, 1\}$ in the discrete cases; on the right: with Transmeta Crusoe $P_{stat} = \frac{44}{57560}$, $\alpha = 3$, and $S = \{0.45, 0.6, 0.8, 0.9, 1\}$ in the discrete cases). Each box aggregates 10 measurements.

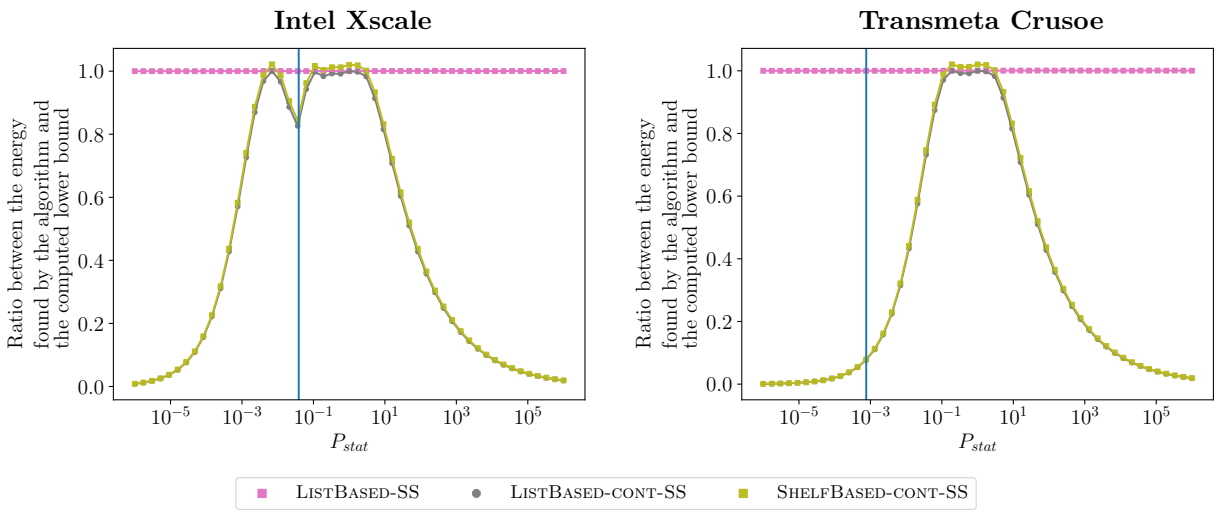

Figure 9: Output energy consumption to compute the solution with discrete and continuous speeds for instances with mixed power and Amdahl's tasks and $p = 32$ processors (on the left: with Intel Xscale $\alpha = 3$, and $S = \{0.15, 0.4, 0.6, 0.8, 1\}$ in the discrete cases; on the right: with Transmeta Crusoe $\alpha = 3$, and $S = \{0.45, 0.6, 0.8, 0.9, 1\}$ in the discrete cases). The vertical plain line corresponds to the actual P_{stat} of the studied processor.

When comparing the Intel Xscale, on the left, to the Transmeta Crusoe, on the right, we see that the available speeds for the Intel Xscale correspond well to the effective P_{stat} (the plain line). It is not the case for the Transmeta Crusoe: lower processor speeds would improve the energy consumption.

9. Conclusion

With the growing concern regarding the energy consumption of current parallel platforms, it is crucial to bound the worst-case performance. This paper is the first to propose such bounds on the energy consumption when scheduling moldable tasks. We highlight the relation between the energy and the completion time (determined by the DVFS mechanism) and rely on the numerous approximation algorithms that have already been proposed to minimize the completion time. This leads to a general mechanism to bound the energy consumption of such existing approximation algorithms for the completion time. In particular, we show that a shelf-based approach is a 3-approximation (resp. 2.08-approximation) algorithm for the energy consumption with discrete speeds (resp. continuous speeds).

We also focus on the optimization of a single shelf by providing a polynomial-time algorithm that can be used to improve existing solutions. Empirical results reveal that such an approach, when combined with a fast optimization post-operation, is beneficial in practice because of its low cost.

To complete this study, we could consider variations of the power model. In particular, we assume that changing frequencies with the DVFS mechanism does not incur any energy cost or delay, which may not be accurate in practice. We additionally assume that the set of available speeds is a constant, while we could consider it a function of the number of processors in use (e.g., when all the processors are used, the highest

frequencies might not be usable). Also, we plan to experiment on more recent processors whose range of frequencies suggest that it behaves closer to the continuous model. This involves doing some measurements to obtain detailed data on the power consumption of such processors and processing these measures, while only information on frequencies is publicly available. Finally, the whole model revolves on the premise that the tasks are computationally bound. In the case of memory bound applications, changing the processor speeds does not change the execution time, which is a case that we have not considered yet. The algorithms presented could be adapted to perform on mixes of computationally bound and memory bound applications.

Overall, this paper aims at providing the theoretical foundations to the problem. Since current task systems do not yet have moldable task profiles that can be used, we have focused on classical models that have already been largely considered in the literature. As soon as moldable task profiles are available, it would be very interesting to conduct experiments on real HPC systems.

Acknowledgements: We would like to thank the anonymous reviewers for their comments and suggestions, which greatly helped improve the paper.

References

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, page 483–485, 1967.

[2] G. Aupy, A. Benoit, and Y. Robert. Energy-aware scheduling under reliability and makespan constraints. In *Proceedings of HiPC'2012*, 2012.

[3] G. Aupy, M. Shantharam, A. Benoit, Y. Robert, and P. Raghavan. Co-scheduling algorithms for high-throughput workload execution. *Journal of Scheduling*, 19(6):627–640, 2016.

[4] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proc. of Int. Parallel and Distributed Processing Symp. (IPDPS)*, pages 113–121, 2003.

[5] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. Embed. Comput. Syst.*, 15(1), 2016.

[6] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):1 – 39, 2007.

[7] A. Benoit, A. Cavelan, V. Le Fèvre, Y. Robert, and H. Sun. A different re-execution speed can help. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, pages 250–257, 2016.

[8] A. Benoit, L. Pottier, and Y. Robert. Resilient co-scheduling of malleable applications. *International Journal of High Performance Computing Applications (IJHPCA)*, 32(1):89–103, 2018.

[9] J. Blazewicz, M. Machowiak, G. Mounié, and D. Trystram. Approximation algorithms for scheduling independent malleable tasks. In *Euro-Par 2001 Parallel Processing*, pages 191–197, 2001.

[10] A. P. Chandrakasan and A. Sinha. JouleTrack: A Web Based Tool for Software Energy Profiling. In *Design Automation Conference*, pages 220–225, Los Alamitos, CA, USA, 2001.

[11] J.-J. Chen and T.-W. Kuo. Multiprocessor energy-efficient scheduling for real-time tasks. In *Proc. of Int. Conf. on Parallel Processing (ICPP)*, pages 13–20, 2005.

[12] E. G. Coffman, Jr, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[13] M. R. Garey and R. L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[15] R. Graham, E. Lawler, J. Lenstra, and A. Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[16] A. Hartstein, V. Srinivasan, T. Puzak, and P. Emma. On the nature of cache miss behavior: Is it $\sqrt{2}$. *Journal of Instruction-Level Parallelism*, 10:1–22, 06 2008.

[17] D. S. Hochbaum and D. B. Shmoys. A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach. *SIAM J. Comput.*, 17(3):539–551, June 1988.

[18] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi. Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster. In *Proc. of the Int. Parallel and Distributed Processing Symposium (IPDPS)*, page 340, 2006.

[19] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of International Symposium on Low Power Electronics and Design (ISLPED)*, pages 197–202, 1998.

[20] J. Keller, S. Litzinger, and C. W. Kessler. Integrating energy-optimizing scheduling of moldable streaming tasks with design space exploration for multiple core types on configurable platforms. *J. Signal Process. Syst.*, 94(9):849–864, 2022.

[21] F. Kong, N. Guan, Q. Deng, and W. Yi. Energy-efficient scheduling for parallel real-time tasks based on level-packing. In *Proc. of the 2011 ACM Symposium on Applied Computing*, pages 635–640, 2011.

[22] R. Krishnamurti and E. Ma. An approximation algorithm for scheduling tasks on varying partition sizes in partitionable multiprocessor systems. *IEEE Transactions on Computers*, 41(12):1572–1579, 1992.

[23] S. Litzinger, J. Keller, and C. Kessler. Scheduling moldable parallel streaming tasks on heterogeneous platforms with frequency scaling. In *2019 27th European Signal Processing Conf. (EUSIPCO)*, pages 1–5. IEEE, 2019.

[24] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC'10: Proc. of the ACM/IEEE Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2010.

[25] T. Okuma, H. Yasuura, and T. Ishihara. Software energy reduction techniques for variable-voltage processors. *Design Test of Computers, IEEE*, 18(2):31 –41, Mar. 2001.

[26] G. Prasanna and B. Musicus. Generalized multiprocessor scheduling and applications to matrix computations. *IEEE Trans. on Parallel and Distributed Systems*, 7(6):650–664, 1996.

[27] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems*, 43:67–80, 2008.

[28] P. Sanders, S. Lamm, L. Hübschle-Schneider, E. Schrade, and C. Dachsbacher. Efficient parallel random sampling - vectorized, cache-efficient, and online. *ACM Trans. Math. Softw.*, 44(3):29:1–29:14, 2018.

[29] E. Saule, D. Bozdag, and Ü. V. Çatalyürek. Optimizing the stretch of independent tasks on a cluster: From sequential tasks to moldable tasks. *J. Parallel Distributed Comput.*, 72(4):489–503, 2012.

[30] H. Sun, R. Elghazi, A. Gainaru, G. Aupy, and P. Raghavan. Scheduling parallel tasks under multiple resources: List scheduling vs. pack scheduling. In *2018 IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)*, pages 194–203, 2018.

[31] J. Turek, J. L. Wolf, K. R. Pattipati, and P. S. Yu. Scheduling parallelizable tasks: Putting it all on the shelf. In *Proc. of the 1992 ACM SIGMETRICS joint Int. Conf. on Measurement and modeling of computer systems*, pages 225–236, 1992.

[32] J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms scheduling parallelizable tasks. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pages 323–332, 1992.

[33] H. Xu, F. Kong, and Q. Deng. Energy minimizing for parallel real-time tasks based on level-packing. In *2012 IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications*, pages 98–103. IEEE, 2012.

[34] H.-E. Zahaf, R. Olejnik, G. Lipari, et al. Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms. *J. of Systems Architecture*, 74:46–60, 2017.