Shelf schedules for independent moldable tasks to minimize the energy consumption

Anne Benoit
LIP, ENS Lyon, France
email: anne.benoit@ens-lyon.fr

Louis-Claude Canon, Redouane Elghazi and Pierre-Cyrille Héam
FEMTO-ST Institute
Univ. Bourgogne Franche-Comté, CNRS, France
email: {louis-claude.canon|redouane.elghazi|pierre-cyrille.heam}@femto-st.fr

***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 moldable tasks that can decide their degree of parallelism at execution (i.e., 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, and simulations are performed to show the performance of the proposed algorithms.**

I. 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 [14], 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 simplest form is already NP-complete (it is indeed identical to 2-Partition [13] 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 [16].

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) [12].

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. Each task in a same shelf starts its 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 to use shelves [28] (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 [17]. More precisely, a processor running at speed s dissipates s^3 watts [18], [25], [9], [4], [10] per time-unit, hence consumes $s^3 \times d$ joules when operated during d units of time. Faster speeds allow for a faster execution, but they also lead to a much higher (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

This work has been supported by the EIPHI Graduate School (contract ANR-17-EURE-0002).

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 be also 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 MinE-Mold-Indep can be solved in polynomial time when processors are independently powered (with or without co-schedules), while the problem becomes NP-complete with simultaneously powered processors (MinE-Mold-Sim).
- We establish multiple approximation ratios for both classical list scheduling algorithms, and shelf-based schedules.
- We provide an optimal dynamic programming algorithm to minimize the energy consumption of a single shelf: 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.

We first discuss related work in Section II. Next, we detail the model (platform, tasks, energy consumption) and schedules, and we introduce the target optimization problems in Section III. The complexity of the problems is established in Section IV. Approximation ratios for MinE-Mold are derived in Section V, and optimal algorithms for a single shelf are provided in Section VI. Finally, a comprehensive empirical study is proposed in Section VII. We conclude and give hints for future research directions in Section VIII.

II. 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. We then proceed on heuristics proposed for real-time systems.

Classical *list scheduling algorithms*, ListBased, constitute the simplest and first heuristic proposed for the rigid case. Tasks are ordered in a priority list and are then scheduled by order of priority, as presented by Garey and Graham [12]: 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 $2 \cdot \max(\frac{W}{p}, t_{\max})$-approximation, where $\frac{W}{p}$ is the average work and $t_{\max}$ is the execution time of the longest task.

Coffman et al. [11] 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 [20], Krishnamurti et al. study a problem similar to ours: 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. Although it is similar when considering a single shelf, it differs from our problem with multiple shelves.

In [28], Turek et al. study the multi-shelves problem. They first offer 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 [28, 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 (which is done in Section VII). ShelfBased is a 3-approximation for the makespan. More precisely, it is a $\left(2 \cdot \frac{W}{p} + t_{\max}\right)$-approximation. To deal with moldable tasks, the authors also present an overall design [28, GF] that works as follows. First, a task and a number of processors are selected. We assume that for a given speed, all tasks will have a shorter execution time than this one. Next, for each other task, we select the number of processors such that the work is minimized. Finally, we solve this instance with rigid tasks and repeat the process for all pairs of tasks and number of processors.

The same year [27], Turek et al. give a 2.7-approximation for the multi-shelves problem, with a fixed number of shelves. However, this algorithm is exponential in the number of shelves.

Aupy et al. tackle the problem of optimizing the power consumption, the makespan and the reliability with dependent non-parallel tasks [2]. They show that most problems are NP-hard and propose heuristics. This paper focuses on moldable tasks.

Finally, many similar works have been proposed in the context of real-time systems with moldable tasks, power constraints and deadlines. The closest considers level-based scheduling (similar to shelves) with rigid or moldable tasks [19]. 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 [29], [30], [21]. In this paper, we do not consider deadlines and we investigate algorithms with guarantees.

III. Model

We first describe the platform model (Section III-A), the task model (Section III-B), the energy model (Section III-C), before formally defining general schedules, single-speed schedules and co-schedules in Section III-D. Finally, we introduce the target optimization problems in Section III-E.

A. 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 $S = \{s_1, s_2, \dots, s_k\}$ 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 [23]. Switching frequencies is usually not allowed during the execution of a given task, but two different tasks scheduled on a same processor can be executed at different frequencies.

B. Tasks

We consider n moldable tasks $\{T_1, T_2, \dots, T_n\}$ with 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 1, 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 [8], [7]).

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

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

C. 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 is $t_{stat}P_{stat}$, 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}). A general model states that the dynamic energy consumption is $t_{dyn}s^{\alpha}$ [5]. Hence, for task T_i, the dynamic energy consumption on each processor is $t_{i,p_i,s_i}s_i^{\alpha}$ (since $t_{dyn} = t_{i,p_i,s_i}$), and the total dynamic energy consumption for the task is $a_{i,p_i,s_i}s_i^{\alpha}$ (the same energy is consumed by each of the p_i processors operating task T_i) with $\alpha > 1$.

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 ($t_{stat} = C_{\max}$).

D. 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 $\delta_i \ge 0$ 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 $T_{i'}$ $(1 \le i, i' \le n$ and $i \ne i')$ 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 W, T_{dyn}, A_{dyn}, A_{stat} and E.

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 I. 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.

E. *Optimization problems*

The general problem is MINE-MOLD: Given n moldable tasks and p processors, the goal is to find a schedule that minimizes the total energy consumption. The processors can either be considered to be simultaneously powered (SIM), or to be independently powered (INDEP). Hence, MINE-MOLD-SIM (resp. MINE-MOLD-INDEP) is the problem with simultaneously-powered (resp. independently-powered) processors.

We also consider the variant of the problem with *rigid* tasks, i.e., when the number of processors per task p_i is fixed (MINE-RIG problem).

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 $s_i \in S$;
- $\sum\limits_{i=1}^{n} p_i \leq p$ (at most p processors are used, since all tasks execute concurrently).

IV. PROBLEM COMPLEXITY

We prove that MINE-MOLD-INDEP can be solved in polynomial time (Section IV-A), while MINE-MOLD-SIM is NP-complete (Section IV-B).

A. *Optimal algorithm for* MINE-MOLD-INDEP

In the *independent* model, the total energy consumption is the sum of the individual energy consumption of each task. We can therefore optimize the energy consumption of each task independently, and execute the tasks one after the other. This is done by executing each task on a single processor, since $(w_{i,j})$ is non-decreasing in j, and hence $a_{i,1,s_i} \leq a_{i,p_i,s_i}$ for all $1 \leq p_i \leq p$. Here, we focus solely on energy optimization, and the time to completion might be very large (a single processor is used).

There is a tradeoff between executing the task fast to reduce the static energy consumption of the task, and running at a slower speed to reduce the dynamic energy consumption. In fact, for each task T_i, we choose the same speed $s_i \in S$ that minimizes $s_i^{\alpha-1} + \frac{P_{stat}}{s_i}$, since the energy consumption of the task is $t_{i,1,s_i}(s_i^{\alpha} + P_{stat})$, and $t_{i,1,s_i} = \frac{w_{i,1}}{s_i}$. Finally, we obtain the optimal value of s_i in $\Theta(|S|)$, by comparing the value for every possible $s_i \in S$.

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 ($A_{stat} = p \times C_{\max}$).

B. *NP-completeness of* MINE-MOLD-SIM

When moving to the *simultaneous* model, it becomes crucial to also minimize the total execution time. We show that the MINE-MOLD-SIM problem actually is NP-complete, even when a single speed is available.

Theorem 1. *The decision problem associated to* MINE-MOLD-SIM *is NP-complete.*

Proof: We first prove that the decision problem associated to MINE-MOLD-SIM 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.

To prove that MINE-MOLD-SIM is NP-hard, we do the reduction from the problem of 3-PARTITION [13]: 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-SIM 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, there is a single speed $S = \{1\}$, and we have $P_{stat} = 1$ and α arbitrarily set to 3.

Therefore, it is always better to execute each task on a single processor, hence leading to 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 $t \times n$. Finally, we set the bound on energy consumption for $\mathcal{I}_2$ to $2nB$.

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 nB, hence a total energy consumption of $2nB$ (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 $2nB$, the total execution time is such that $t \leq B$, and the sum of a_i's in each subset S_j cannot exceed B. Therefore, $\mathcal{I}_1$ has a solution, which concludes the proof. ■

V. Approximation ratios for MinE-Mold-Sim

To solve MinE-Mold-Sim, we extend a strategy [28, 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 such as ListBased or ShelfBased. We thus start by considering the rigid case. Moreover, we first consider a simplified version of the problem where all tasks have the same speed in Section V-A, before moving to the general case in Section V-B.

A. 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:

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

with $\alpha > 1$.

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 $(M_i, \rho s_\lambda, \frac{\delta_i}{\rho})$, 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 $K_{[\lambda]} = s_\lambda \times pC_{\max}(\lambda)$. 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 $s_{\rho\lambda} = \rho \times s_\lambda$.

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.

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

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

$$\begin{aligned} E(\lambda) &= W s_\lambda^{\alpha-1} + A_{stat}(\lambda) \times P_{stat} & (1)\\ &= W s_\lambda^{\alpha-1} + C_{\max}(\lambda) \times p \times P_{stat}. & (2)\end{aligned}$$

The problem can be split as two decisions to take:

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

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

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

hence proving the lemma. ■

Note that if λ^* is a single-speed schedule minimizing the makespan (necessarily $s_{\lambda^*} = s_{\max}$), then for any single-speed schedule λ with $s_\lambda = s_{\max}$, $C_{\max}(\lambda^*)s_{\max} \leq C_{\max}(\lambda)s_{\max}$. It follows that $K_{[\lambda^*]} \leq K_{[\lambda]}$.

Let us denote by $\lambda_{\mathcal{A}}$ the schedule returned by $\mathcal{A}$. Let $s^*_{[\lambda_{\mathcal{A}}]} \in S$ be a speed for which $\min_{\lambda \in [\lambda_{\mathcal{A}}]} E(\lambda)$ is attained. Let λ^{OPT} be a single-speed schedule minimizing the energy. From Equation (2), we have:

$$E^{\text{OPT}} = W s_{\lambda^{\text{OPT}}}^{\alpha-1} + pC_{\max}(\lambda^{\text{OPT}}) \times P_{stat}.$$

By definition of λ^*, $E^{\text{OPT}} \geq W s_{\lambda^{\text{OPT}}}^{\alpha-1} + pC_{\max}(\lambda^*) \times P_{stat}$. Now, by optimality of $s^*_{[\lambda_{\mathcal{A}}]}$ and since $K_{[\lambda_{\mathcal{A}}]} \leq cK_{[\lambda^*]} \leq cK_{[\lambda^{\text{OPT}}]}$ and $c \geq 1$:

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

thus proving the theorem. ■

2) Moldable case: We present the overall design of the algorithm MINE-MOLD-SIM:

- First, we select one task $T_{i'}$, a number of processors $p_{i'}$ for this task, and we assume that for any given speed s, all tasks will have an execution time at most $t_{\max} \triangleq t_{i',p_{i'},s}$ ($t_{i,p_i,s} \leq t_{i',p_{i'},s} = t_{\max}$ for $1 \leq i \leq n$). There are np such assumptions to consider.
- For each value of $(T_{i'}, p_{i'})$, 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: $p_i = \arg\min_{1 \leq j \leq p} jt_{i,j}$ such that $t_{i,p_i} \leq t_{\max}$.

Intuitively, we analyze the approximation ratio of any moldable scheduling algorithm with the following approach based on [28, GF]:

- For a given $t_{\max}$, we bound the cumulative work to be executed assuming all task execution duration is bounded by $t_{\max}$.
- We then bound the maximum makespan achievable with a heuristic for MINE-RIG-SIM.
- Finally, we bound the maximum total energy consumption during this duration.

We can state the main result of this section.

Theorem 3. *In the moldable single-speed context (*MINE-MOLD-SIM*), we assume that there exists a polynomial-time algorithm $\mathcal{A}$ for* MINE-RIG-SIM *that returns a schedule $\lambda_{\mathcal{A}}$ such that $K_{[\lambda_{\mathcal{A}}]} \leq a \cdot \frac{W([\lambda_{\mathcal{A}}])}{p} + b \cdot t_{\max}(\lambda_{\mathcal{A}})$ (resp. $K_{[\lambda_{\mathcal{A}}]} \leq \max\left(a \cdot \frac{W([\lambda_{\mathcal{A}}])}{p}, b \cdot t_{\max}(\lambda_{\mathcal{A}})\right)$), where a and b are constants satisfying $a + b \geq 1$. One can compute in polynomial time a schedule $\lambda_\bullet$ such that $\lambda_\bullet$ running at a speed that minimizes the energy consumption consumes at most $a + b$ (resp. $\max(a, b)$) times the optimal energy.*

Proof: The proof is done for $K_{[\lambda_{\mathcal{A}}]} \leq a \cdot \frac{W([\lambda_{\mathcal{A}}])}{p} + b \cdot t_{\max}(\lambda_{\mathcal{A}}) \cdot s_{\lambda_{\mathcal{A}}}$. The other case (max of the two terms instead of sum) is similar.

Let λ^{OPT} be a schedule minimizing the energy (for MINE-MOLD-SIM with a single speed).

Let $G = \left\{\frac{w_{i,p'}}{p'} \mid 1 \leq i \leq n,\ 1 \leq p' \leq p\right\}$.

For any task i and any number $k \leq m$ of processors, we denote by $\lambda_{i,k}$ the schedule returned by $\mathcal{A}$ on the (rigid) instance: for all i', $p_{i'}$ is the integer in $\{1, \ldots, p\}$ minimizing $w_{i',\ell}$ under the constraint $\frac{w_{i',p_\ell}}{p_\ell} \leq \frac{w_{i,k}}{k}$. There are at most pn different $\lambda_{i,k}$. Let $\lambda_\bullet$ be a schedule minimizing the $E(\lambda_{i,k})$.

Denoting by i^{OPT} and $p_{i^{\text{OPT}}}$ respectively the task and the number of processors such that $t_{\max}(\lambda^{\text{OPT}}) = t_{i^{\text{OPT}},p_{i^{\text{OPT}}}}$, one has by construction of the $\lambda_{i,k}$, $W(\lambda_{i^{\text{OPT}},p_{i^{\text{OPT}}}}) \leq W(\lambda^{\text{OPT}})$. Set $\lambda_{g^{\text{OPT}}} = \lambda_{i^{\text{OPT}},p_{i^{\text{OPT}}}}$.

Running $\lambda_{i,k}$ at a speed $s_{i,k} \in S$ that minimizes the energy, we have

$$\begin{aligned}
E(\lambda_\bullet) \leq E(\lambda_{g^{\text{OPT}}}) &\leq E(\frac{s_{\text{OPT}}}{s_{\lambda_{g^{\text{OPT}}}}}\lambda_{g^{\text{OPT}}}) \\
&\leq W(\lambda_{g^{\text{OPT}}})^{\alpha-1} + \frac{K_{[\lambda_{g^{\text{OPT}}}]}}{s_{\text{OPT}}} \\
&\leq (W^{OPT})^{\alpha-1} + (a+b) \cdot \frac{K_{[\lambda^{\text{OPT}}]}}{s_{\text{OPT}}} \\
&\leq (a+b) \cdot (W^{OPT})^{\alpha-1} + (a+b) \cdot \frac{K_{[\lambda^{\text{OPT}}]}}{s_{\text{OPT}}} \\
&\leq (a+b) \cdot E(\lambda^{\text{OPT}}),
\end{aligned}$$

which concludes the proof. ■

Recall that LISTBASED is an algorithm with a max and $a = 2$ and $b = 2$ [12], hence it is a 2-approximation algorithm. SHELFBASED, an algorithm with a sum and $a = 2$ and $b = 1$ [28, LTF], is thus a 3-approximation algorithm.

B. 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.

Theorem 4. *In the moldable context (*MINE-MOLD-SIM*), we assume that there exists a polynomial-time algorithm $\mathcal{A}$ for* MINE-RIG-SIM *that returns a schedule $\lambda_{\mathcal{A}}$ such that $C_{\max}(\lambda_{\mathcal{A}}) \leq a \cdot \frac{A_{dyn}(\lambda_{\mathcal{A}})}{p} + b \cdot t_{\max}(\lambda_{\mathcal{A}})$ (resp. $C_{\max}(\lambda_{\mathcal{A}}) \leq \max\left(a \cdot \frac{A_{dyn}(\lambda_{\mathcal{A}})}{p}, b \cdot t_{\max}(\lambda_{\mathcal{A}})\right)$), where a and b are constants satisfying $a \geq 1$. One can compute in polynomial time a schedule $\lambda_\bullet$ such that: $E(\lambda_\bullet) \leq (a+b)E^{\text{OPT}}$ (resp. $E(\lambda_\bullet) \leq \max(a, b+1)E^{\text{OPT}}$).*

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

Let $G = \{t_{i,p',s} \mid 1\ \leq i \leq n,\ 1 \leq p' \leq p,\ s \in S\}$. Note that G can be computed in polynomial time. For a schedule λ, set $X_i(\lambda) = s_i^\alpha a_{k,p_i,s_i} + P_{stat} a_{k,p_i,s_i}$.

For any i, p', s we consider the rigid instance defined by: $p_{i'}, s_{i'}$ in order to minimize $s_{i'}^\alpha a_{i',p_{i'},s_{i'}} + P_{stat} a_{i',p_{i'},s_{i'}}$ 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 $\lambda_{i,p',s}$. There are a polynomial number of such schedules, and each one can be computed in polynomial time.

Let $\lambda_\bullet$ be a scheduled among the $\lambda_{i,p',s}$ minimizing the energy. Let λ^{OPT} be a schedule minimizing the energy for the main problem. Let also i^{OPT}, $s_{i^{\text{OPT}}}$ and $p_{i^{\text{OPT}}}$ satisfying $t_{\max}(\lambda^{\text{OPT}}) = t_{i^{\text{OPT}},p_{i^{\text{OPT}}},s_{i^{\text{OPT}}}}$.

By construction, one has:

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

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

$$\begin{aligned}
E(\lambda_\bullet) &\leq E(\lambda_{i^{\text{OPT}},p_{i^{\text{OPT}}},s_{i^{\text{OPT}}}}) = E(\lambda_g) \\
&\leq \sum_i X_i(\lambda_g) + P_{stat} \cdot p \cdot C_{\max}(\lambda_g) \\
&\leq \sum_i X_i(\lambda^{\text{OPT}}) + P_{stat} \cdot p \cdot C_{\max}(\lambda_g) \\
&\leq \sum_i X_i(\lambda^{\text{OPT}}) + P_{stat} \cdot p \cdot \max(a \cdot \frac{A_{dyn}}{p}, b \cdot t_{\max}(\lambda_g)) \\
&\leq \max(\sum_i X_i(\lambda^{\text{OPT}}) + aP_{stat}A_{dyn}, \\
&\qquad \sum_i X_i(\lambda^{\text{OPT}}) + bpP_{stat}t_{\max}(\lambda_g)) \\
&\leq \max(a\sum_i X_i(\lambda^{\text{OPT}}) + aP_{stat}A_{dyn}, \\
&\qquad E^{\text{OPT}} + bpP_{stat}C_{\max}(\lambda^{\text{OPT}})) \\
&\leq \max(aE^{\text{OPT}}, (b+1)E^{\text{OPT}}) \\
&\leq \max(a, b+1)E^{\text{OPT}},
\end{aligned}$$

which concludes the proof. ■

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

VI. 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. Formally, given a set of n tasks and p processors, the goal is to find an assignment $((p_i),(s_i))_{1\leq i\leq n}$ that minimizes $E = \sum_{i=1}^{n} \left(p_i t_{i,dyn} s_i^{\alpha} + p_i t_{i,stat} P_{stat}\right)$, where

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

A. Preliminaries

Since the static energy spent depends on the total length of the shelf ($\max_{1\leq i\leq n} t_{i,p_i,s_i}$), the algorithm proceeds by fixing the shelf length to T, and aims at finding the optimal number of processors and speed for each task, such that the time bound T 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 T, we consider the function $OptS(w, T)$ that returns the optimal speed such that $\frac{w}{s} \leq T$ and the energy consumption $ws^{\alpha-1} + TP_{stat}$ is minimized. Since the energy consumption is an increasing function of s for $s \geq 0$, the optimal speed is the smallest speed such that the shelf length is not exceeded. Therefore, $OptS(w, T) = \min\left\{s \in S \mid s \geq \frac{w}{T}\right\}$. In the case no such speed exists, the function returns None.

Note that this function can be computed in $\Theta(\log(|S|))$ by doing a binary search within values of S.

B. Optimal algorithm for MinE-OneShelf-Sim

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 $T \in \mathcal{T}$, we compute the solution for each set of tasks $T_1, \ldots, T_i$, $i \in \llbracket 1, n \rrbracket$ and each number of processors $q \in \llbracket 1, p \rrbracket$.

Let $e_{i,q}$ be the minimum energy consumed by task T_i on q processors, while not exceeding time T. It is computed by using the function $OptS(t_{i,q}, T)$, 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 $1 \leq i \leq n$ and $1 \leq q \leq p$ as:

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

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 1 provides the corresponding pseudo-code of this dynamic programming algorithm.

Theorem 5. MinE-OneShelf-Sim *can be solved optimally in polynomial time.*

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

Base case: $\forall q \in \llbracket 0, p \rrbracket$, the energy consumed to handle no task on q processors is 0, meaning that the $E_{0,q}$ values for $q \in \llbracket 0, p \rrbracket$ are correct.

Inductive step: Let $i \in \llbracket 0, n-1 \rrbracket$, and we assume that $\forall q \in \llbracket 1, p \rrbracket$, $E_{i,q}$ 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 $k \in \llbracket 1, q-i \rrbracket$, 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 $O(n^2p^3|S|)$. ■

VII. Empirical Study

A. Experimental setup

All the base algorithms rely on the global mechanism presented in Section V-A2 [28, GF] with either a single speed or multiple speeds. It is then combined with the strategies presented in Section II: ListBased and ShelfBased (for a total of four heuristics). 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 VI (this may change the number of processors used for each task);
- 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. When both optimizations are selected, OptiShelf would be performed first but we discarded this combination because it did not outperform De-shelf in practice. This represents a total of ten heuristics.

To compare the different heuristics, we implemented them in C++17 compiled with `gcc` 9.3.0 with optimization option `-O3`.

Algorithm 1: Optimal algorithm for MinE-OneShelf-Sim

```
$\mathcal{T} \leftarrow \emptyset$ ;
for $i \leftarrow 1$ to $n$ do
    for $q \leftarrow 1$ to $p$ do
        for $s \in S$ do
            $\mathcal{T} \leftarrow \mathcal{T} \cup \{\frac{t_{i,q}}{s}\}$ ;
$res \leftarrow \infty$ ;
for $T \in \mathcal{T}$ do
    for $i \leftarrow 1$ to $n$ do
        for $q \leftarrow 1$ to $p$ do
            $s_{i,q} \leftarrow OptS(t_{i,q}, T)$ ;
            if $s_{i,q} =$ None then
                $e_{i,q} \leftarrow \infty$ ;
            else
                $e_{i,q} \leftarrow q t_{i,q} s_{i,q}^{\alpha-1} + qTP_{stat}$ ;
    for $q \leftarrow 0$ to $p$ do
        $E_{0,q} \leftarrow 0$ ;
    for $i \leftarrow 1$ to $n$ do
        for $q \leftarrow i$ to $p$ do
            for $k \leftarrow 1$ to $q - i + 1$ do
                if $E_{i-1,q-i} + e_{i,k} < E_{i,q}$ then
                    $E_{i,q} \leftarrow E_{i-1,q-k} + e_{i,k}$ ;
    if $E_{n,p} < res$ then
        $res \leftarrow E_{n,p}$
return $res$ ;
```

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].

B. Instance generation

The following characteristics were extracted from a realistic platform [22], [6]: $p = 32$ Intel Xscale with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$.

The number of tasks varies from 20 to 1000, with a step every 20 tasks. The workload was generated as follows:

- half of the task profiles are generated with Amdahl's law [1] [26] ($w_{i,p_i} = w_{i,1} \cdot \beta + \frac{w_{i,1} \cdot (1-\beta)}{p_i}$, where $w_{i,1}$ and β are chosen through uniform distribution $U(0,1)$);
- half of the task profiles are generated with the Power law [24] [15] [26] ($w_{i,p_i} = \frac{w_{i,1}}{p_i^{\beta}}$ where $w_{i,1}$ and β are chosen through uniform distribution $U(0,1)$).

C. 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 IV-A). In that case, the static energy is paid only while a task is executed, and any solution to MinE-Mold-Sim will consume at least as much energy as this lower bound.

Figure 1 presents an overview of the results for all heuristics, for the case with $n = 500$ tasks. We report both the ratio between the energy consumption of each heuristic with the lower bound (the lower the better), and also the execution time of the heuristics. For convenience, SS refers to the single-speed variants, and MS to the multiple-speed ones. A first remark is that single-speed and multiple-speed variants give very similar results. Indeed, in practice, the multiple-speed heuristics give the same speed to most tasks.

In Figure 2, we compare the most efficient heuristic, ListBased-MS, with the fastest one, ShelfBased-SS. We can see that ListBased-MS provides schedules with lower energy than ShelfBased-SS, but at the cost of a much larger execution time for the algorithm[2].

In Figure 3, we 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. With this comparison, 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

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

[2] ListBased could be implemented in a faster way with a segment tree to compute which task can be started, making this operation $\log n$ instead of n. However, this complex data structure would probably not be included in most implementations.

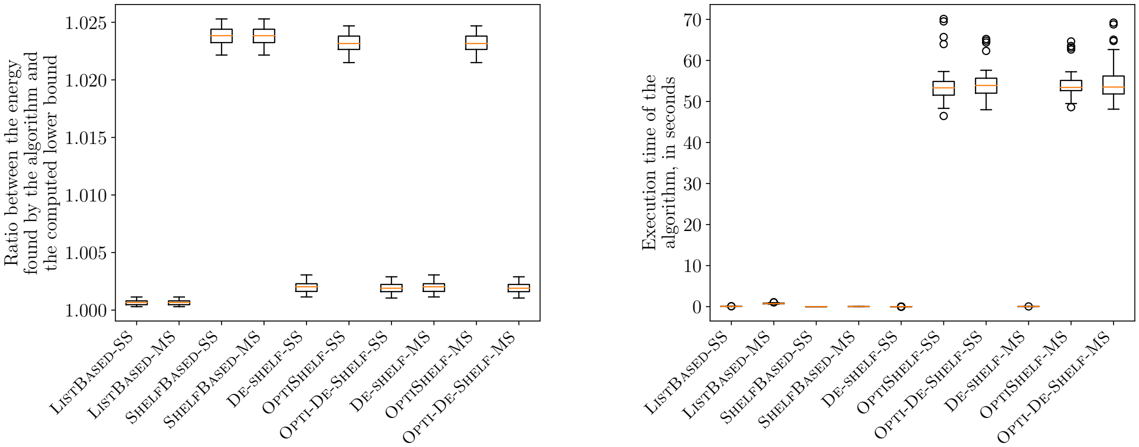

Figure 1. Output energy consumption and execution time to compute the solution for all ten heuristics with $n = 500$ mixed power and Amdahl's tasks and $p = 32$ processors (with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$).

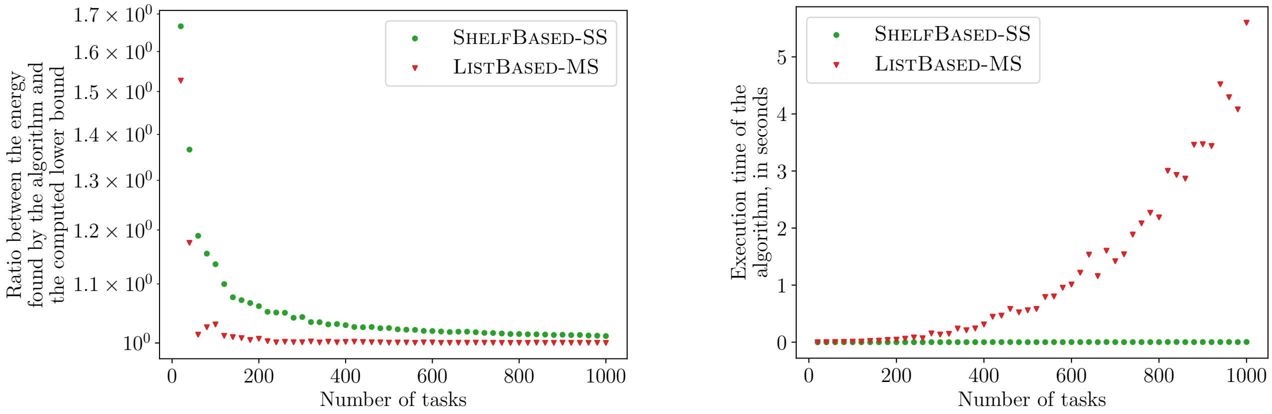

Figure 2. Output energy consumption and execution time to compute the solution for ListBased-MS and ShelfBased-SS for instances with mixed power and Amdahl's tasks and $p = 32$ processors (with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$).

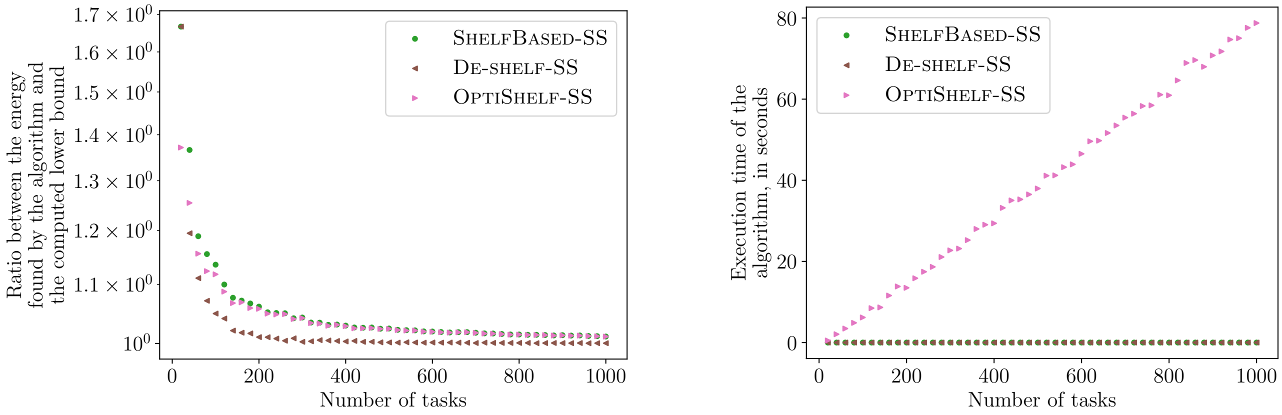

Figure 3. Output energy consumption and execution time to compute the solution for ShelfBased-SS and optimizations (OptiShelf and De-shelf) for instances with mixed power and Amdahl's tasks and $p = 32$ processors (with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$).

execution time. On the other hand, the overhead of De-shelf is small, allowing to get a solution of better quality at a small cost.

Finally, in Figure 4, we compare ListBased with multiple speeds to ShelfBased with a single speed, with

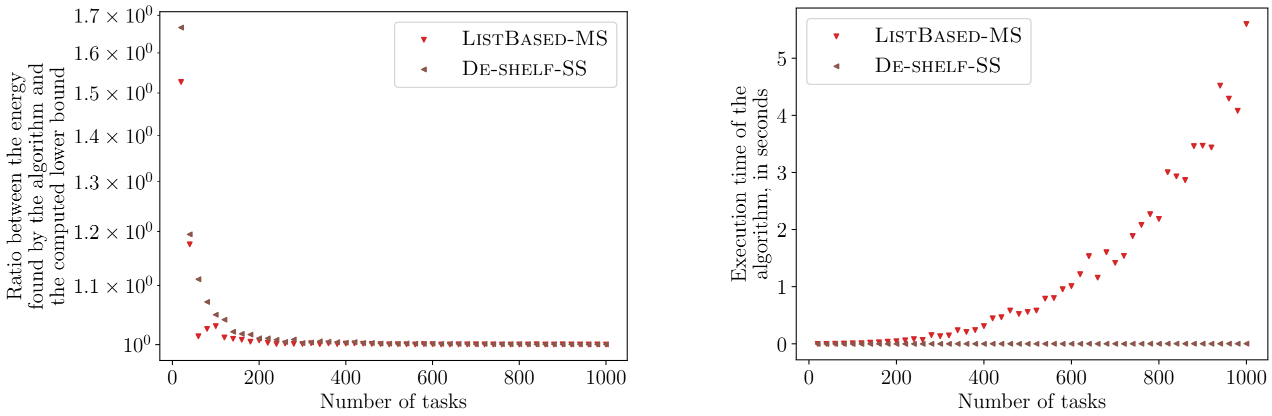

Figure 4. Output energy consumption and execution time to compute the solution for ListBased-MS and De-shelf-SS for instances with mixed power and Amdahl's tasks and $p = 32$ processors (with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$).

De-shelf, which we found to be the best optimization for ShelfBased. For small instances, ListBased still performs better than ShelfBased with De-shelf. However, when n grows larger, ShelfBased with De-shelf performs as well as ListBased, but with a lower time complexity.

VIII. 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 bound 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, which 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 algorithm for the energy. Empirical results reveal that such an approach, when combined with a fast optimization post-operation, is beneficial in practice because of its low cost. This paper also contributes to the optimization of a single shelf by providing a polynomial algorithm.

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. Also, we assume that the set of available frequencies is finite and discard continuous frequencies even though they may provide insights or lead to optimal solutions, which may then be used as a basis for the finite case.

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.

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] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.