Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Optimising makespan and energy consumption in task scheduling for parallel systems

Russell Stewart[a], Andrea Raith[a], Oliver Sinnen[b,*]

[a] Engineering Science, University of Auckland, New Zealand
[b] Electrical, Computer, and Software Engineering, University of Auckland, New Zealand

ARTICLE INFO

Keywords:
Task scheduling
Parallel processing
Bi-objective
Mixed-integer linear program
Pareto-efficient solutions

ABSTRACT

In parallel computing, the scheduling of the tasks of an application onto the processors of the parallel system is crucial. A task schedule determines both the allocation of tasks to the processors, and the order in which they are executed. Formally defined, this task scheduling problem is a challenging optimisation problem (strongly NP-hard), even for the case where there is only one objective, e.g. to minimise the total execution time, also known as makespan. Today task scheduling is often a constrained optimisation problem, where the makespan needs to be minimised, while keeping energy consumption below a threshold, or vice versa, where the energy consumption needs to be minimised, while keeping the makespan below a threshold. In a generalisation of this problem, we consider here a bi-objective version of this scheduling problem that aims to minimise both makespan and energy consumption in a Pareto-efficient manner. Based on a model of processor energy consumption, a bi-objective integer linear programming problem is formulated. Different approaches to modelling processor static (background) energy consumption are described. The task scheduling problems can be solved using bi-objective optimisation methods based on weighted sum scalarisation or ε-constraint scalarisation. In an extensive evaluation, the computational performance and the characteristics of sets of Pareto-efficient solutions of this bi-objective problem are studied and discussed, with interesting insights into the nature and shape of the Pareto-efficient sets.

1. Introduction

Current processor technology reached a physical limit more than a decade ago and the maximum processing frequency has not increased significantly since then (Olukotun and Hammond, 2005). Further improvements of computing power are now achieved by parallelisation, i.e. the use of multiple processors for one software program. To efficiently use a parallel computer consisting of multiple processors (or cores), a program needs to be broken down into tasks (Grama et al., 2003). These tasks are then allocated and ordered for execution on the processors. This process is referred to as scheduling and is crucial to achieve a high performance execution of a program (Drozdowski, 2009; Sinnen, 2007).

In scheduling theory and computing practice (Sinnen, 2007; Augonnet et al., 2011; Cojean et al., 2019), a program to be executed on a parallel system is represented as a Directed Acyclic Graph (DAG) called a task graph, where the nodes represent the tasks and the edges represent communications or dependencies between them. Weights assigned to the nodes and edges represent the work to be processed and data communication time, respectively. Given a task graph and a number of target processors, the objective of task scheduling is to find a processor and start time for every task, so that a certain criterion is optimised, for example the total execution time of the task graph, called makespan. This problem is a well known NP-hard optimisation problem (Rayward-Smith, 1987), even for many special cases. As a consequence, many heuristics have been proposed that provide good, but not optimal solutions (Drozdowski, 2009; Sinnen, 2007). Unfortunately, no α-approximation algorithm is known, hence there is no guarantee for the quality of these schedules. Despite the NP-hardness, there has been an effort to investigate and propose optimal scheduling algorithms for small to medium sized problem instances (Davidović et al., 2007; Sinnen, 2014; Venugopalan and Sinnen, 2015). Being able to compute optimal schedules can be very useful for highly time critical systems, or where an execution schedule is repeated many times. Equally important, it allows an assessment of the quality of heuristic solutions and might help to gain insights into the nature of optimal solutions, which can further inform development of heuristics.

While the makespan or schedule length is a very important criterion for the scheduling of a program, there has been an increased interest

* Corresponding author.
E-mail addresses: rste933@aucklanduni.ac.nz (R. Stewart), a.raith@auckland.ac.nz (A. Raith), o.sinnen@auckland.ac.nz (O. Sinnen).

https://doi.org/10.1016/j.cor.2023.106212
Received 14 June 2022; Received in revised form 2 February 2023; Accepted 1 March 2023
Available online 3 March 2023

in other, sometimes secondary optimisation criteria. One of the most important ones is the energy consumption of a parallel system during the execution of the program. Modern processors are able to change the execution frequency of each processor during the execution of a program. Lower frequencies allow the processor to also reduce the supply voltage, which both in turn reduce the power consumption. As a consequence, the relation between power consumption and execution frequency can be approximated as quadratic, which implies that slower execution of a task can save energy (Aupy et al., 2013; Colin et al., 2016; Xie et al., 2022; Zhu et al., 2004).

Using this technique to reduce energy consumption, a large number of scheduling algorithms have been proposed that consider the energy consumption in scheduling algorithms for parallel computers, e.g. Benoit et al. (2013), Chen et al. (2022), Hu et al. (2019), Liu et al. (2023), Wang et al. (2010), just to name a few. In the overwhelming majority of the works (Xie et al., 2022), the energy consumption is minimised under a performance constraint (e.g. a given makespan threshold) or the best schedule is searched for a given energy budget. Few heuristics have been proposed that optimise both criteria at the same time e.g. Ahmad et al., 2008; Lee and Zomaya, 2010; Pillai et al., 2018, and we are not aware of any work that does bi-objective scheduling in a Pareto-efficient manner, i.e. the equivalent of optimal scheduling in the bi-objective case.

Again, it would be desirable to have good quality solutions as references, in particular those that are Pareto-efficient. Being able to find sets of Pareto-efficient solutions would give insight into the nature of such sets and might advise users of parallel systems how to most efficiently use the multiple processors with a parallel program. For example, whether to use many processors with slow frequency or few with high frequency.

We attempt to fill this research gap by proposing a bi-objective Mixed Integer Linear Program (MIP) for this scheduling problem that generates Pareto-efficient solutions. It is based on a previous MIP formulation for the single-objective (makespan) scheduling problem (Venugopalan and Sinnen, 2015) and integrates dynamic power consumption models (Aupy et al., 2013). In addition, we further extend the MIP to also integrate different static power consumption models (i.e. power consumption of idle processors). By doing so, the paper makes the following contributions:

- Proposal of the first Pareto-efficient task scheduling approach, based on a MIP formulation, with the two objectives to minimise makespan and energy consumption.
- Integration of different realistic static energy consumption models with appropriate extension of the MIP.
- Extensive experimental evaluation, giving insights into the nature and shape of Pareto-efficient solutions sets for this bi-objective version of a classical scheduling problem.

The rest of the paper is organised as follows. Section 2 formally defines the scheduling problem, in particular the task graph and the parallel system. This is followed by a detailed study of the related work of optimal task scheduling and scheduling that considers energy. Section 4 then proposes the bi-objective MIP. How to solve this MIP in order to obtain a set of Pareto-efficient solutions, is discussed in Section 5. We perform an evaluation of the proposed solution approach in Section 6 using a diverse set of task graphs, processor numbers and available frequency settings, and discuss results and insights in Section 7. The paper concludes in Section 8.

2. Problem definition

Let us start by formally defining the scheduling problem that is addressed here.

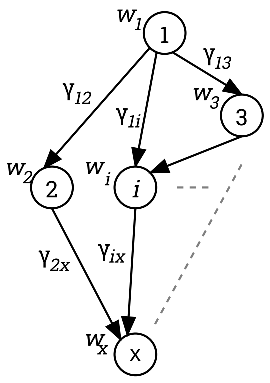


Fig. 1. Example task graph.

2.1. Task graph

The program or application to be scheduled is represented by a directed acyclic graph, or task graph $G = (\mathcal{V}, \mathcal{E}, w, \gamma)$. The nodes $\mathcal{V}$ in the graph represent the application's tasks, and node weights w define the amount of work (corresponding to time depending on the processor speed) required to process each task. The directed edges $\mathcal{E}$ in the graph represent the communication between tasks and imply precedence: the task from which an edge leaves must be completed before commencing the task an edge enters. Finally, edge weights γ specify the communication time necessary if two tasks, connected by an edge, are allocated to *different* processors. Fig. 1 shows an example illustrating the above definition of the task graph.

In summary, the following notation is used:

$\mathcal{V}$	=	Set of tasks , $\mathcal{V} = \{1, \dots, \lvert\mathcal{V}\rvert\}$.
$\mathcal{E}$	=	Set of edges where $(i,j) \in \mathcal{E}$ if $i,j \in \mathcal{V}$ and task i must be completed before task j begins.
w_i	=	Computational work of task i for all $i \in \mathcal{V}$, $w_i \in \mathbb{N}$.
γ_{ij}	=	Communication time penalty for tasks i and j for all $(i,j) \in \mathcal{E}$, $\gamma_{ij} \in \mathbb{N}_0$. This only applies if two tasks with a precedence relation are processed on different processors.

For convenience, we define the set of predecessors of a task $j \in \mathcal{V}$ as $\delta^-(j)$:

$$\delta^-(j) \quad = \quad \{i \in \mathcal{V} \mid (i,j) \in \mathcal{E}\}.$$

2.2. Parallel system

Task graph G is to be scheduled on a parallel system consisting of a set of processors $\mathcal{R}$. A fully connected communication network links these processors, communication is performed by a communication subsystem (i.e. processors are not involved) and there is no contention for communication resources. All processors are homogeneous in their functional capability (all processors are equally capable of executing each of the tasks), but they can run at different execution speeds.

Aupy et al. (2013) survey energy consumption models for parallel processors and algorithms to minimise energy consumption. This work is based on the ability of modern processors to perform dynamic voltage and frequency scaling (DVFS). Hence, to save energy, the voltage and frequency of a processor can be scaled down (implying slower execution). While their work considers different approaches to scaling, e.g. at fixed intervals or during the execution of a task, we concentrate on the practical and realistic variant where the frequency remains unchanged during the execution of a task but it can change between every task.

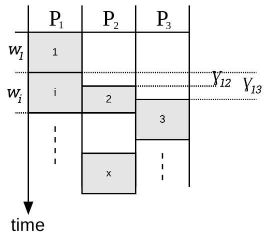


Fig. 2. Example schedule (without speed scaling) for graph of Fig. 1.

We also assume that the processors have a limited number of discrete speeds or frequencies available. Following the classical power consumption model of the literature (Pruhs et al., 2008), we assume that the power consumption of a processor is proportional to the square of its execution speed/frequency. Note, that it would also be possible to define a power rating for each discrete processor speed (i.e. a mapping rather than a proportional value) to further generalise the power consumption model. While this would only lead to minimal changes in the MIP proposed below, we omit this here due to the lack of meaningful processor data.

Hence we define the parameters of the target parallel system as follows:

$\mathcal{R}$	=	Set of processors, $\mathcal{R} = \{1, 2, \dots, \lvert\mathcal{R}\rvert\}$
$\mathcal{S}$	=	Set of processor speeds, $\mathcal{S} = \{s_1, s_2, \dots, s_{\lvert\mathcal{S}\rvert}\}$ with $s_i \in \mathbb{N}, s_1 < s_2 < \cdots < s_{\lvert\mathcal{S}\rvert}$

2.3. Scheduling problem

For a task graph and a parallel system as defined in the previous two subsections, our goal is to find a schedule for all tasks $\mathcal{V}$ on the processors of the parallel system $\mathcal{R}$. Finding a schedule corresponds to allocating each task i to a processor $r_i \in \mathcal{R}$, assigning it a start time t_i and choosing a processor speed from $\mathcal{S}$ for its execution.

The two scheduling objectives are to minimise the makespan (i.e. the finish time of the last task) and to minimise the energy consumption for the execution of this schedule on the parallel system.

For a schedule to be valid, two constraints have to be fulfilled for all tasks according to the defined model. As the assumption is a non-preemptive execution, tasks' executions cannot overlap if allocated to the same processor:

$$r_i = r_j \Rightarrow t_i + d_i \le t_j \vee t_j + d_j \le t_i \quad \forall i, j \in \mathcal{V}, \tag{1}$$

with d_i being the task duration, i.e. the speed-scaled computation work of task i (more on this in Section 4). And to adhere to the precedence constraints, we must have for every edge $(i, j) \in \mathcal{E}$

$$t_j \ge t_i + d_i + \begin{cases} \gamma_{ij} & \text{if } r_i \neq r_j \\ 0 & \text{if } r_i = r_j \end{cases} \tag{2}$$

Fig. 2 illustrates a valid example schedule for the task graph of Fig. 1. In this case there is no speed scaling, i.e. all tasks are assumed to be executed at speed 1. Observe that task i, depending on task 1 can start immediately after 1 finishes (communication costs are neglected on the same processor), but tasks 2 and 3 are delayed by the respective communication times between the tasks γ_{12} and γ_{13}.

3. Related work

After the definition of the problem in the previous section, let us now look at related work. This can be roughly divided into two categories: *(i)* task scheduling that considers energy consumption, as discussed in Section 3.1; and *(ii)* optimal task scheduling, which is presented in Section 3.2.

3.1. Energy considering scheduling algorithms

3.1.1. Machine and flowshop scheduling

Before we look specifically at algorithms for task scheduling on parallel computing systems, let us briefly digress to the related field of parallel machine and flowshop scheduling approaches that consider energy consumption. In many cases the energy consumption model of the relevant works is different to what is addressed in this paper. Works by Li et al. (2016), Wang et al. (2018) for example, assume that the energy consumption is heterogeneous across machines, i.e. different machines have different energy consumption rates. In other work, the energy price varies with time and an objective is to minimise the monetary energy cost (Anghinolfi et al., 2021). Other algorithms consider energy, but focus on the peak energy consumption rather than the total energy consumption (Fang et al., 2013). The work by Wu and Che (2019) is more closely related to our work here as it uses speed-scaling, like the DVFS used here, to change the processing speed of machines and hence to optimise the energy consumption. It uses also the same type of algorithmic approaches that are used in task scheduling for parallel computing systems, like list scheduling and meta-heuristics.

3.1.2. Task scheduling for parallel systems

In scheduling for parallel systems, execution performance (e.g. makespan, throughput etc.) is naturally the major optimisation goal (and often the only one). When energy consumption is also considered, we can distinguish between three scenarios:

1. optimise energy under performance constraint
2. optimise performance under energy constraint
3. optimise energy and performance simultaneously

The majority of the proposed scheduling algorithms in literature that consider energy target the first two scenarios (Xie et al., 2022), while this paper proposes an approach to optimise both performance (makespan) and energy at the same time (scenario 3).

Many different algorithmic approaches are used in all scenarios in the literature, mostly domain-specific heuristics, meta-heuristics or machine learning approaches. Apart from performance and energy, these algorithms might consider further criteria, e.g. reliability (Huang et al., 2018) and real-time constraints (Hu et al., 2019), but we focus here only on the former criteria. By design, these approaches usually obtain good, but not optimal, solutions for the overall optimisation problem, even though in some works aspects of the problem are solved optimally within an overarching heuristic approach.

Examples for algorithms targeting the first scenario are Liu et al. (2023), Safari and Khorsand (2018), Zhang et al. (2018). The approaches of Liu et al. (2023), Zhang et al. (2018) belong to a prominent category of algorithms that consist of two phases. In the first phase a schedule is obtained that optimises the performance goal, e.g. makespan. Then in the second phase DVFS is used two slow down the execution of tasks that have 'slack', i.e. their slower execution has no or a limited negative impact on the performance. This so-called *slack reclamation* lowers the energy consumption, while maintaining the performance (at the optimal level or within a given requirement).

In the recent and extensive survey of scheduling algorithms for parallel systems that consider energy consumption by Xie et al. (2022), the authors list more than 30 recent algorithms falling into the first category.

In the second scenario the energy consumption is limited or constrained and the algorithms try to optimise the performance, often paired with further criteria, like reliability. Again many different algorithmic approaches are employed (mostly domain specific heuristics and meta-heuristics), where Xie et al. (2022) list more than a dozen recent ones. Examples of such algorithms are Chen et al. (2022), Quan et al. (2019), Xie et al. (2019).

Optimising both makespan and energy consumption at the same time has been mostly neglected so far (Xie et al., 2022, only references half a dozen papers), e.g. Ahmad et al. (2008), Lee and Zomaya (2010), Pillai et al. (2018).

Regarding the use of integer/mixed integer linear programming as we use in this work, there have only been very few proposed algorithms (Eitschberger and Keller, 2020; Hu et al., 2019; Huang et al., 2018; Qin et al., 2019; Zhou et al., 2019). Among those five algorithms, four are scheduling with real-time constraints (Hu et al., 2019; Huang et al., 2018; Qin et al., 2019; Zhou et al., 2019), while Eitschberger and Keller (2020), Hu et al. (2019), Huang et al. (2018), Qin et al. (2019) are targeting the first optimisation scenario (optimising energy under performance constraints) and Zhou et al. (2019) targets scenario two.

In terms of the scheduling model, Eitschberger and Keller (2020) is closest to our work presented here, but as stated, they are not applying bi-objective optimisation to find sets of Pareto-efficient solutions as proposed here.

To summarise, the relevant MIP-based scheduling algorithms that consider energy are listed in the upper part of Table 1.

3.2. Optimal scheduling algorithms

Almost all of the algorithms we have referred to above do generally not guarantee to obtain optimal solutions as they are based on heuristics, meta-heuristics and machine learning. For single-objective task scheduling with makespan minimisation a very small number of optimal algorithms have been proposed.

In a rough categorisation, approaches that compute optimal solutions for the task scheduling problem on parallel systems can be divided into state–space search based approaches and mathematical programming based approaches. Examples for state–space search are Dietze and Rünger (2020), Davidović and Crainic (2015), Orr and Sinnen (2020, 2021), which are usually based on local search, depth-first search branch-and-bound or A*.

Mathematical programming, as used in this paper, has been applied by Davidović et al. (2007), Mallach (2018), Roy et al. (2019a,b, 2020), Tang et al. (2017, 2020), Venugopalan and Sinnen (2015) to optimally solve the scheduling problem. Most of these articles propose MIP formulations, with the exception of Malik et al. (2017/2018), where a Satisfiability Modulo Theory (SMT) approach is taken.

While some of the above research works have considered slightly different scheduling models (e.g. considering processor heterogeneity, processor networks or allowing task duplication etc.), their common ground is the focus on the optimisation of a single criterion — makespan.

To summarise, these MIP-based optimal scheduling algorithms are listed in the lower part of Table 1.

From this literature review we conclude that there is a clear research gap for the third optimisation scenario, the simultaneous optimisation of both makespan and energy, as only heuristic based approaches exist so far. We address this gap by proposing a *bi-objective* MIP formulation with the aim of finding Pareto-efficient solutions that is adapted from a single-objective MIP formulation (Venugopalan and Sinnen, 2015). Our approach is not a heuristic approach, it is the bi-objective equivalent to optimal scheduling in the single-objective case. Our approach allows to find *all* Pareto-efficient solutions for a given (small) problem instance.

4. Model formulation

In this section we develop and propose a bi-objective MIP to solve the scheduling problem defined in Section 2. After the model presentation we propose important static energy extensions to the base model. The two objectives to minimise are the makespan and the energy consumption.

4.1. Base model decision variables

The base model presented below is based on the model presented in Venugopalan and Sinnen (2015). There they define an optimal task scheduling MIP for homogeneous, fixed-speed, multi-processor systems, minimising the makespan of the schedule.

Because in the model proposed by Venugopalan and Sinnen (2015) all assumed processors run at a uniform and constant speed, task work and task duration (i.e. the execution time) are indistinguishable quantities in the formulation: the optimal schedule would be the same regardless of whether task work or duration was modelled. With variable processor speeds, the actual execution time (d_i) of task i is proportional to the task work (w_i) and selected processor speed (s). Assuming a constant processor speed is maintained during the processing of task i, then the relationship between these quantities is simply: $d_i = \frac{w_i}{s}$.

A solution to the scheduling problem defined in Section 2 is a valid allocation of a processor, start time and duration (equivalent to the allocation of a processor speed) to each task $i \in \mathcal{V}$ as defined Section 2 and repeated here.

r_i	=	Assigned processor for task i
t_i	=	Scheduled start time of task i
d_i	=	Duration of task i

In order to obtain an MIP model for which solvers can obtain a solution quickly (Venugopalan and Sinnen, 2015), the following binary variables are introduced for all $i, j \in \mathcal{V}, i \neq j$:

$$\sigma_{ij} = \begin{cases} 1 & \text{task } i \text{ finishes before task } j \text{ starts} \\ 0 & \text{otherwise} \end{cases}$$

$$\epsilon_{ij} = \begin{cases} 1 & \text{processor of task } i \ (r_i) \text{ is less than that of task } j \\ 0 & \text{otherwise} \end{cases}$$

The variables σ_{ij} will be used to enforce precedence constraints induced by the edges. And in combination with the ϵ_{ij} variables, the MIP can enforce that the execution of tasks allocated to the same processor is not overlapping.

A binary decision variable maps tasks to discrete speeds (which determines task duration):

$$z_{ik} = \begin{cases} 1 & \text{if task } i \text{ uses speed } s_k \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \mathcal{V}, \forall k \in \{1, \dots, |S|\}$$

The model objectives are defined using schedule length or makespan (W) and energy consumption (E) decision variables. The dynamic energy ($E_{dynamic}$) is the total energy required to process all tasks. In the base formulation, only dynamic energy is considered, thus static energy is set to zero, $E_{static} = 0$. We introduce and discuss static energy models in Section 4.4.

4.2. Base bi-objective model

The following bi-objective MIP aims to minimise both the schedule makespan and energy consumption (A1). Constraints (A2a)–(A2c) define the components of the objectives. Constraints (A3)–(A5) ensure tasks only overlap if assigned to different processors (adhering to the processor constraint (1)), (A6)–(A7) deal with processor assignment, and (A8)–(A10) enforce task precedence (adhering to the precedence constraint (2)) (Venugopalan and Sinnen, 2015). Lastly, constraints (A11) and (A12) define processor speeds and derive task duration.

An upper bound on the makespan, W_{max}, is obtained by assuming all tasks run at the lowest speed on the same processor, $W_{max} = \sum_{i \in \mathcal{V}} \frac{w_i}{s_1}$. It is used as a big-M value in (A8).

Table 1
Closely related MIP-based scheduling algorithms and their optimisation objectives.

Reference	Scenario	Performance	Energy	Model variation
Energy considering algorithms				
Eitschberger and Keller (2020)	1	constraint	optimised	
Hu et al. (2019)	1	constraint	optimised	real-time
Huang et al. (2018)	1	constraint	optimised	real-time, reliability
Qin et al. (2019)	1	constraint	optimised	real-time
Zhou et al. (2019)	2	optimised	constraint	security
Optimal algorithms				
Davidović et al. (2007)	–	optimised	–	
Mallach (2018)	–	optimised	–	
Roy et al. (2019a,b, 2020)	–	optimised	–	comm. contention
Tang et al. (2017, 2020)	–	optimised	–	task duplication
Venugopalan and Sinnen (2015)	–	optimised	–	
This paper	3	optimised	optimised	*(Pareto-efficient solutions)*

$$\min \begin{pmatrix} W \\ E \end{pmatrix} \tag{A1}$$

$$\text{s.t.} \quad t_i + d_i \le W \qquad \forall i \in \mathcal{V} \tag{A2a}$$

$$\sum_{i\in\mathcal{V}} \sum_{k=1}^{|\mathcal{S}|} z_{ik} s_k^2 w_i = E_{dynamic} \tag{A2b}$$

$$E = E_{dynamic} + E_{static} \tag{A2c}$$

$$\sigma_{ij} + \sigma_{ji} \le 1 \qquad \forall i \ne j \in \mathcal{V} \tag{A3}$$

$$\epsilon_{ij} + \epsilon_{ji} \le 1 \qquad \forall i \ne j \in \mathcal{V} \tag{A4}$$

$$\sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} \ge 1 \qquad \forall i \ne j \in \mathcal{V} \tag{A5}$$

$$r_j - r_i - 1 - (\epsilon_{ij} - 1)|\mathcal{R}| \ge 0 \qquad \forall i \ne j \in \mathcal{V} \tag{A6}$$

$$r_j - r_i - \epsilon_{ij}|\mathcal{R}| \le 0 \qquad \forall i \ne j \in \mathcal{V} \tag{A7}$$

$$t_i + d_i + (\sigma_{ij} - 1)W_{max} \le t_j \qquad \forall i \ne j \in \mathcal{V} \tag{A8}$$

$$t_i + d_i + \gamma_{ij}(\epsilon_{ij} + \epsilon_{ji}) \le t_j \qquad \forall j \in \mathcal{V} : i \in \delta^-(j) \tag{A9}$$

$$\sigma_{ij} = 1 \qquad \forall j \in \mathcal{V} : i \in \delta^-(j) \tag{A10}$$

$$\sum_{k=1}^{|\mathcal{S}|} z_{ik} \frac{w_i}{s_k} = d_i \qquad \forall i \in \mathcal{V} \tag{A11}$$

$$\sum_{k=1}^{|\mathcal{S}|} z_{ik} = 1 \qquad \forall i \in \mathcal{V} \tag{A12}$$

$$\sigma_{ij} \in \{0,1\} \qquad \forall i \ne j \in \mathcal{V} \tag{A13}$$

$$\epsilon_{ij} \in \{0,1\} \qquad \forall i \ne j \in \mathcal{V} \tag{A14}$$

$$r_i \in \{1,\dots,|\mathcal{R}|\} \qquad \forall i \in \mathcal{V} \tag{A15}$$

$$t_i \ge 0 \qquad \forall i \in \mathcal{V} \tag{A16}$$

$$W \ge 0 \tag{A17}$$

$$z_{ik} \in \{0,1\} \qquad \forall i \in \mathcal{V},\ \forall k \in \{1,\dots,|\mathcal{S}|\} \tag{A18}$$

In comparison to the single-objective MIP formulation of Venugopalan and Sinnen (2015) we have introduced heterogeneity, in terms of processor speeds, and integrated the second optimisation criterion to minimise the energy consumption. We achieved this with careful choices that keep the formulation as simple as possible and close to the original MIP. To that end, a new binary variable z_{ik}, the duration of execution d_i and the constraints (A2b), (A2c), (A11), (A12), (A18) were introduced.

4.3. Ensuring integrality of time variables

From the existence of the different processor speeds it follows that the execution duration d_i of the task i can vary, with $d_i = \frac{w_i}{s_k}$ when executed with speed $s_k \in \mathcal{S}$. A processor speed $s_k \in \mathcal{S}$ is modelled as a small integer, representing a multiple of a base frequency unit. For instance, $\mathcal{S} = \{3,6,9\}$ with base frequency unit of $333\frac{1}{3}$ MHz, corresponding to processor frequencies of 1 GHz, 2 GHz and 3 GHz.

While the task weight w_i and the speeds s_k are integers, the duration d_i is in general a rational number. Consequently, the start and finish times of the tasks can also be rational numbers, represented as floating point numbers. With the MIP's use of the variables σ_{ij} and ϵ_{ij} to avoid task overlaps and to enforce precedence constraints, floating point precision problems can lead to difficulties, e.g. when ensuring that the start time of one task is greater than the finish time of another. For better solvability of the MIP formulation and to avoid precision problems with floating point numbers, it is desirable to ensure that the durations are also integers, so that in turn all task starting and finish times are also integers within the MIP model.

When the weights of the task weights and the processor speeds of a problem instance do not ensure this naturally, the task weights can be scaled up. Ideally, the smallest possible scaling constant is chosen to avoid the (scaled) schedule makespan objective from becoming too large. The scaling method used is as follows.

A simple solution is to scale, i.e. multiply, all task weights $w_i \in \mathcal{V}$ by the product of all processor speeds, $\prod_{s_k\in\mathcal{S}} s_k$. This will guarantee that all task execution durations are natural numbers. Note that the same scaling also needs to be done with the edge weights, which represent a time delay, to keep the consistency between the edge weights and the task weights. To avoid creating task weight values that are too large, especially for many large speed values, it is possible to lower this scaling factor with the greatest common divisor of the processor speed set $\mathcal{S}$, respectively excluding one speed.

$$SF = \frac{\prod_{s_k\in\mathcal{S}} s_k}{gcd\left\{\prod_{s_k\in\mathcal{S}\,:\,s_k\ne s_h} s_k \,:\, s_h \in \mathcal{S}\right\}}$$

By definition, SF is a natural number, as all $s_k \in \mathcal{S}$ are natural numbers. Scaling all task weights $w_i \in \mathcal{V}$ and all edge weights $\gamma_{ij} \in \mathcal{E}$ with this factor SF guarantees that all durations and in turn all start and finish times are natural numbers. The scaling can be done as a pre-processing step or integrated into the MIP formulation.

4.4. Static energy

In this section, we explore three types of static energy models for E_{static}. This is beyond the general accounting for static energy as is commonly done in algorithms that consider energy (e.g. Colin et al., 2016; Xie et al., 2022). Here, we also examine the case where there is no idle power at all (and in turn idle energy consumption) due to the ability of modern cores/processors to be turned off/put to sleep (Arora et al., 2015). Fig. 3 illustrates the three different models we consider, where grey boxes represent tasks being executed and blue background colour indicates idle time where static energy is being considered.

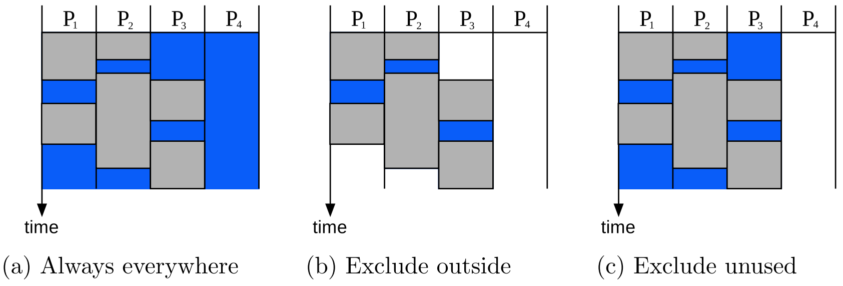


Fig. 3. Schedule illustrations of static energy models. Grey boxes: tasks with dynamic & static energy; blue background: idle period with static energy; white background: idle, no static energy.

The *always everywhere* static model (Fig. 3(a)) assumes that there is a constant static load, present on each processor, for the entire schedule makespan. This is true during the execution of tasks as well as during all idle periods. In the *exclude outside* model (Fig. 3(b)), a processor q can switch off (no static load is applied) before processing the first task assigned to q and after processing the last task assigned to q. The *exclude unused* static model (Fig. 3(c)) assumes that a static load is only applied to processors that are assigned one or more tasks: any processor not used for the full duration of a schedule can be turned off to avoid static energy consumption.

The static energy models require three additional parameters. $\mathcal{D}$ is the set of all dummy tasks. These act as pointers to processors, with one dummy task assigned to each processor q, where $u_q \in \mathcal{D}$ defines the dummy task assigned to processor $q \in \mathcal{R}$. It is worth noting that the dummy tasks have no associated task start time or duration (i.e. variables t_{u_q} and d_{u_q} are not defined in the model). Lastly, the parameter ω is the static power value.

For each processor $q \in \mathcal{R}$, two decision variables measure the duration of the opening and closing processor idle times:

ϕ_q	=	Initial idle time on processor q: from start of model period up until start of first task scheduled on processor q.
ψ_q	=	Closing idle time on processor q: from completion of last task assigned to processor q, until end of task schedule.

A binary decision variable indicates whether a processor $q \in \mathcal{R}$ is utilised during the duration of the schedule:

$$\mu_q = \begin{cases} 1 & \text{if at least one task is scheduled on processor } q \\ 0 & \text{otherwise} \end{cases}$$

Below, the constraints required for each of the three static models are discussed. Unless stated otherwise, each static model also requires all constraints from the base model given in Section 4.2.

4.4.1. Always everywhere static energy

Under the *always everywhere* model, the total static energy consumption (E_{static}) is simply the product of the background static power (ω), the total number of processors ($|\mathcal{R}|$), and the schedule makespan (W):

$$E_{static} = \omega|\mathcal{R}|W \tag{A19a}$$

4.4.2. Exclude outside static energy

Constraint (A19b) redefines static energy consumption and is used in place of (A19a).

$$E_{static} = \omega|\mathcal{R}|W - \omega\sum_{q\in\mathcal{R}}(\phi_q + \psi_q) \tag{A19b}$$

The domains of a number of existing constraints must be extended to include both the set of real tasks ($\mathcal{V}$) and the set of dummy tasks ($\mathcal{D}$). In constraints (A4), (A6), (A7), (A14) and (A15) $\forall i \neq j \in \mathcal{V}$ is replaced by $\forall i \neq j \in \mathcal{V} \cup \mathcal{D}$.

Constraints (A20) to (A24) are added to define the length of time that processor q is idle at the beginning and end of the schedule.

$$\phi_q - \epsilon_{u_q i}W_{max} - \epsilon_{iu_q}W_{max} - t_i \le 0 \quad \forall i \in \mathcal{V},\ \forall q \in \mathcal{R} \tag{A20}$$

$$\psi_q - \epsilon_{u_q i}W_{max} - \epsilon_{iu_q}W_{max} - W + (t_i + d_i) \le 0 \quad \forall i \in \mathcal{V},\ \forall q \in \mathcal{R} \tag{A21}$$

$$r_{u_q} = q \quad \forall q \in \mathcal{R} \tag{A22}$$

$$\phi_q + \psi_q - W \le 0 \quad \forall q \in \mathcal{R} \tag{A23}$$

$$\phi_q,\ \psi_q \ge 0 \quad \forall q \in \mathcal{R} \tag{A24}$$

In constraints (A20) and (A21) $\epsilon_{u_q i} = \epsilon_{iu_q} = 0$ for the set of tasks i in $\mathcal{V}$ that are assigned to the same processor as dummy task u_q. In (A20), this will set ϕ_q to the earliest start time of this set of tasks: giving the initial idle time on processor q. Similarly, in (A21), this will set ψ_q to the shortest duration between the completion of each task in this set, and the end of the schedule: defining the closing idle time on processor q. Constraint (A22), along with the domain extended constraints listed above, assign each dummy task the correct processor index. Lastly, (A23) is necessary when no tasks are assigned to processor q, and sets the upper bound for the sum of the idle times (ϕ_q and ψ_q) to the schedule duration W.

4.4.3. Exclude unused static energy

This static energy consumption model can be considered a sub-case of the previous *exclude outside*. In this model, only a processor $q \in \mathcal{R}$ that is fully idle (i.e. no task is assigned to it, $\mu_q = 0$) can be turned off, hence does not consume static energy. As a sub-case, the MIP model uses the same static energy equation as the previous model, namely (A19b), and all constraints from the *exclude outside* model. To enforce that only on fully idle processors the static energy consumption is zero, we add the following constraints:

$$\mu_q W_{max} + \epsilon_{u_q i}W_{max} + \epsilon_{iu_q}W_{max} - d_i \ge 0 \quad \forall i \in \mathcal{V}\ \ \forall q \in \mathcal{R} \tag{A25}$$

$$(1 - \mu_q)W_{max} - \phi_q - \psi_q \ge 0 \quad \forall q \in \mathcal{R} \tag{A26}$$

$$\mu_q \in \{0, 1\} \quad \forall q \in \mathcal{R} \tag{A27}$$

Constraint (A25) determines processor utilisation. When at least one task is assigned to processor q, it follows that there will exist a task $i \in \mathcal{V}$ such that $\epsilon_{u_q i}W_{max} + \epsilon_{iu_q}W_{max} = 0$. Because of this, the processor q utilisation binary variable μ_q will be forced to take the value 1 to satisfy constraint (A25) (given that there is no zero task durations). With $\mu_q = 1$, constraint (A26), will force the (non-negative) idle time decision variables (ϕ_q and ψ_q) to value 0.

Alternatively, if no tasks are assigned to processor q, μ_q can take the preferred value 0, and still satisfy constraint (A25). In turn, the initial and closing processor idle time variables will not be forced to take value 0. They will be maximised by the solver to $\phi_q + \psi_q = W$, limited by (A23), as this minimises the static energy (A19b).

The consideration of static energy in its different models is a novel contribution to MIP formulations for our scheduling problem. We achieved this with the introduction of the decision variables ϕ, ψ and μ and the additional constraints (A19a) to (A27).

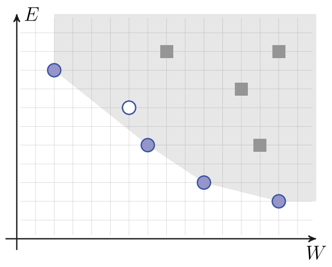


Fig. 4. Points with different (W, E): Square points are dominated, round points are non-dominated, where filled points are supported, and hollow points are non-supported. The convex hull of feasible objective vectors is shown in grey.

5. Solving bi-objective task scheduling problem

The variants of the scheduling problems described in Sections 4.2 and 4.4 have the two objectives to minimise makespan W and energy consumption E (A1) making the problem a linear bi-objective optimisation problem with integer variables, i.e. bi-objective MIPs. To solve this bi-objective problem means to identify all solutions (schedules) that are Pareto-efficient (Ehrgott, 2005). Given two schedules with different makespan and energy consumption (W_1, E_1) and (W_2, E_2) the first schedule is said to *dominate* the second one if $W_1 \leq W_2$ and $E_1 \leq E_2$ with at least one strict inequality. A schedule is called *Pareto-efficient* if there exists no other schedule that dominates it with respect to (W, E). The objective vector (W, E) of a Pareto-efficient schedule is called *non-dominated*.

The aim is to identify efficient solutions of the problem so that one efficient solution per non-dominated vector (W, E) is obtained. For an integer MO problem two types of efficient solutions are distinguished: *Supported* solutions can be obtained as optimal solutions of a (single-objective) weighted sum problem with objective

$$\min \lambda W + (1 - \lambda)E \tag{3}$$

for some $\lambda \in (0, 1)$ whereas *non-supported* efficient solutions cannot be computed this way (Ehrgott, 2005). Fig. 4 illustrates the different types of solutions, and their corresponding (W, E)-values. The non-dominated points that correspond to supported efficient solutions lie on the lower-left boundary of the convex hull of the set of feasible objective vectors (shown in grey), as these are exactly the solutions that can be obtained by a weighted sum problem (3), whereas non-supported solutions lie in the interior of the convex hull.

5.1. Solving bi-objective MIPs

In order to solve bi-objective MIPs exactly there are two available strategies, namely scalarisation or the application of a bespoke bi-objective algorithm. Halffmann et al. (2022) provide an excellent overview of the literature.

A scalarisation (see overview in Eichfelder, 2008) turns a bi-objective or multi-objective problem into a related single-objective problem that can then be solved with standard MIP solvers. A well-known scalarisation-based method is the dichotomic method (e.g. Cohon et al., 1979; Aneja and Nair, 1979) that solves a series of problems where the two objectives are replaced by a single weighted-sum objective (details in Section 5.2). The ε-constraint scalarisation (Chankong and Haimes, 1983) retains a single objective and turns the other objective(s) into constraints, see also Section 5.3. Some scalarisations, such as ε-constraint scalarisation have the disadvantage that they affect problem structure and can make the resulting optimisation problems more challenging to solve (e.g. Ehrgott and Ryan, 2002). While weighted-sum scalarisation retains problem structure, making scalarised problems as easy to solve as the single-objective version of the problem, it can only identify a subset of efficient solutions. Scalarisations can be applied within so-called image space decomposition approaches that provide different strategies to explore a multi-objective problem's non-dominated solutions in objective space (e.g. Hamacher et al., 2007; Boland et al., 2017; Doğan et al., 2021).

On the other hand, the most common type of bespoke algorithms extend the branch-and-bound or branch-and-cut framework for MIPs to the bi-objective and multi-objective case. This is challenging as bounding is much weaker in the multi-objective case, and therefore the developed algorithms can be computationally expensive even for specific problem classes and small problem instances (e.g. Stidsen et al., 2014; Gadegaard et al., 2019; Parragh and Tricoire, 2019). Implementations of these algorithms are generally not available.

Therefore, we apply two different image space decomposition approaches to be able to take advantage of fast standard MIP solvers while guaranteeing efficient solutions are obtained. As shown in the computational evaluation, the dichotomic method (Section 5.2) is relatively fast but only finds a subset of solutions, whereas the box method (Section 5.3) is slower, thereby limiting the size of problems that can be tackled, but it guarantees to obtain a complete set of efficient solutions.

5.2. Dichotomic method

In the following, we apply the so-called dichotomic method that iteratively computes solutions of (3) for varying weights λ e.g. Cohon et al., 1979; Aneja and Nair, 1979. Initially two schedules are computed that are lexicographically optimal, by choosing a small value $\delta > 0$ and solving the problem with objective (3) once with $\lambda = 1 - \delta$ giving a solution with objective vector (W_1, E_1) that minimises W and once with $\lambda = \delta$ giving a solution with objective vector (W_2, E_2) that minimises E as illustrated in Fig. 5(a). The initial values of λ are not set to 1 and 0, respectively, to ensure lexicographic solutions are found, i.e. solutions with minimal values of W and E, respectively, that are not dominated. In the next iteration the problem is solved again with objective (3) and $\lambda^* = \frac{E_1 - E_2}{W_2 - W_1 + E_1 - E_2}$ yielding an optimal solution with objective vector (W^*, E^*). The search direction of the weighted objective function of this weighted sum problem is indicated by an arrow in Fig. 5(a). Two cases can occur.

1. If $\lambda^* W^* + (1 - \lambda^*)E^* < \lambda^* W_1 + (1 - \lambda^*)E_1$ we next solve two new optimisation problems with new weights λ^* calculated as above for the pair of points (W_1, E_1) and $(W_2, E_2) = (W^*, E^*)$, and the pair of points $(W_1, E_1) = (W^*, E^*)$ and (W_2, E_2) as shown in Fig. 5(b).
2. Alternatively, if $\lambda^* W^* + (1 - \lambda^*)E^* = \lambda^* W_1 + (1 - \lambda^*)E_1 = \lambda^* W_2 + (1 - \lambda^*)E_2$ then no more supported solutions with better objective value with respect to λ^* exist with $W_1 \leq W \leq W_2$ and $E_1 \geq E \geq E_2$ and no further optimisation problems are generated. This happens for the right-most weighted sum problem in Fig. 5(b).

This process continues (Fig. 5(c)) until no more problems need to be solved. It is guaranteed that at least one supported efficient solution is found for all $\lambda \in (0, 1)$. It should be noted that applying the weighted sum scalarisation (3) limits our results to obtaining only *supported* efficient solutions.

5.3. Box method

In order to compute all efficient solutions, including non-supported ones, a different scalarisation of the problem is necessary. For instance, the ε-constraint method (Chankong and Haimes, 1983) or one of its variants such as the box-method (Hamacher et al., 2007) can be applied. In the box method, initially the lexicographic solutions are computed. Together they define the upper left and lower right corners of a box in which all other non-dominated points must be situated as shown in Fig. 6(a). The box method then splits this box in half along one of the objectives (we choose W here), limits the feasible region, and identifies another non-dominated solution. This is done by

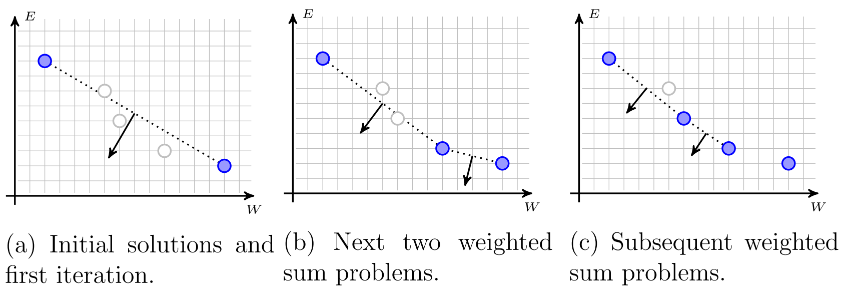

E
W
(a) Initial solutions and first iteration.
(b) Next two weighted sum problems.
(c) Subsequent weighted sum problems.

Fig. 5. Illustration of dichotomic method: First steps.

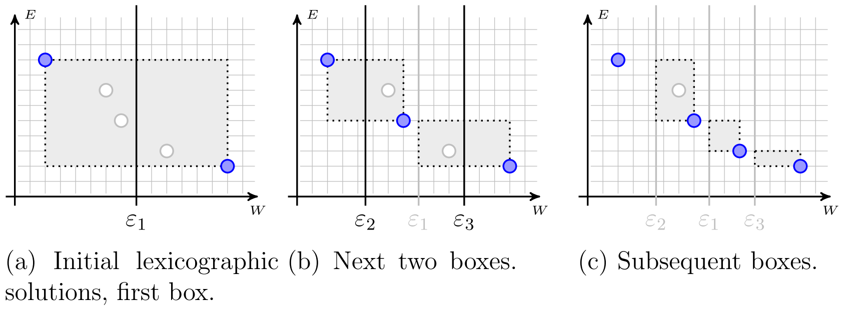


Fig. 6. Illustration of the box method.

solving a so-called ε-constrained optimisation problem where one of the two objectives (E in the example) becomes the optimisation problem's single objective, and the other one (W in the example) is limited by ε with the inclusion of an additional constraint:

$$\begin{aligned} \min \quad & E \\ \text{s.t.} \quad & W \le \varepsilon \\ & \text{all other constraints.} \end{aligned} \tag{4}$$

In Fig. 6(a), the first choice of $\varepsilon = \varepsilon_1$ is indicated, and the non-dominated point obtained by solving (4) is shown in Fig. 6(b). For every new point found, boxes can be split and are thus iteratively updated to limit the area in objective space where more non-dominated points may be situated. This continues until boxes reach an area deemed sufficiently small. Figs. 6(b) and (c) show subsequent iterations (choices of ε and boxes) for the example. For further details, the reader is referred to Hamacher et al. (2007).

The box method can identify all non-dominated points, including non-supported ones, which generally comes at the expense of computational effort, as the discussion in Section 7 shows. We chose the box method here, rather than a conventional ε-constraint approach, as it explores areas in objective space in a manner that ensures that larger "gaps" between non-dominated points are explored first thereby giving a good overview of the available efficient solutions even if the method is terminated early due to runtime limitations.

6. Evaluation

Proposals of new algorithms or MIP formulations for a parallel computing scheduling problem usually conduct a comparative study of the proposed approaches with the state-of-the-art in the field. This work is novel as it proposes MIP formulations and applies methods for the computation of sets of Pareto-efficient solutions for the bi-objective task scheduling problem. To the best of our knowledge, there are no comparable works.

Previous studies on *single* objective task scheduling optimisation (Davidović et al., 2007; Venugopalan and Sinnen, 2016; Orr and Sinnen, 2020) have shown that a number of the graph characteristics have an effect on solution times, apart from the input size, i.e. number of tasks. An example is the edge density or graph structure. Because this was studied before, we instead focus here on the novel bi-objective and energy aspects. Our evaluation aims to answer questions related to this, as follows.

Are the proposed methods useful to compute Pareto-efficient sets? For this we look at the solution times and the number of obtained solutions, distinguishing between the MIP formulations and the solution methods. Is one of the solution methods superior to the other?

What are the shapes of the sets of Pareto-efficient solutions? The shape of these sets has not been studied and characterised before and can give important insights into which static energy model is meaningful and into the available trade-offs found by the two solution methods. Do the processor speed sets have an impact on the shapes?

6.1. Methodology

To answer those questions we implemented the here proposed bi-objective MIP formulation (Section 4) with the four different static energy models **none**, **always everywhere**, **exclude outside** and **exclude unused** (Section 4.4). The MIPs were formulated using the open source Python optimisation library PuLP 1.6, the two methods were also implemented in python, and the weighted sum and ε-constraint scalarisations were solved with Gurobi 8. All experiments were executed on our lab PCs with Intel Core i7 quad core processor, 3.4 GHz, and 16 GB of main memory running Windows 10.

A diverse set of task graphs (Section 6.2.1) were generated and scheduled on different numbers of processors and speed sets with each of the MIP formulations. To find a set of Pareto-efficient solutions for each problem instance, made up of a task graph and a target system, we employed the two proposed solution methods (Section 5): the **dichotomic** method, which searches for a set of supported, efficient solutions; the **box** method, which can locate well distributed supported and non-supported efficient solutions.

We implemented the dichotomic weighted sum (box) method in such a way that the adjacent pair of non-dominated points with maximum distance (the box with maximum area) are selected for further exploration in each step. This allows us to obtain a subset of non-dominated points with a good spread across all non-dominated points, if the method is terminated early due to time limitations.

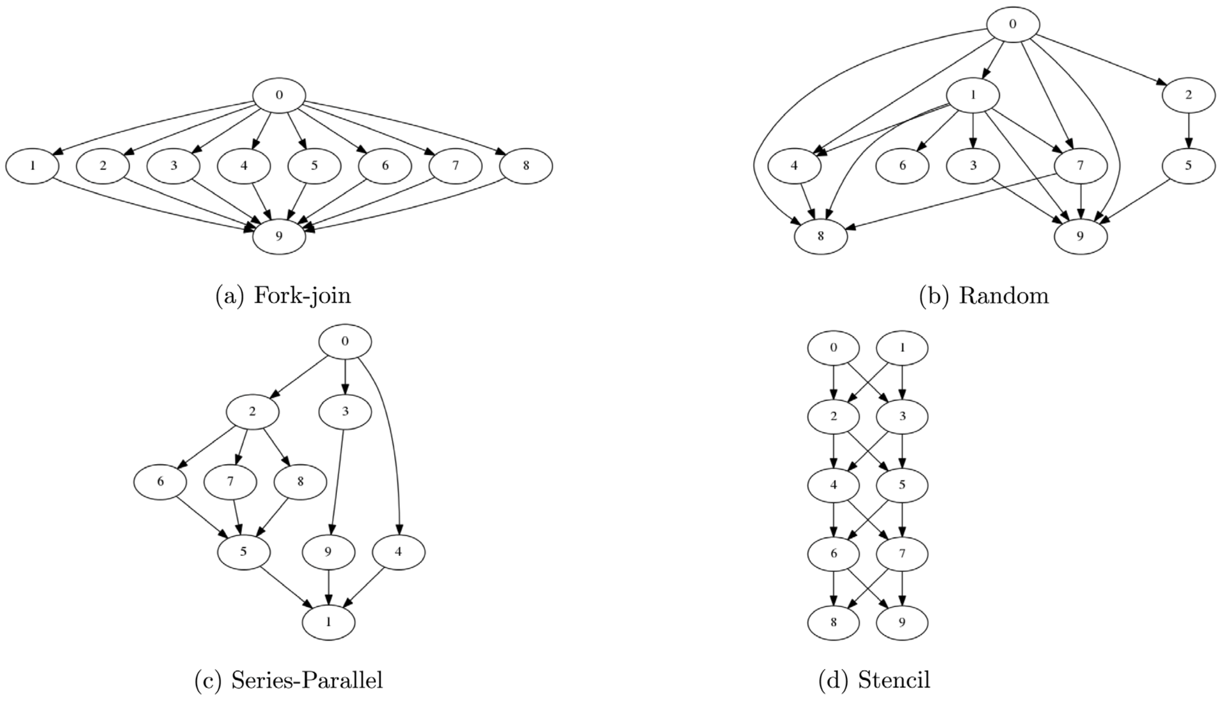


Fig. 7. Illustration examples of used graph structures.

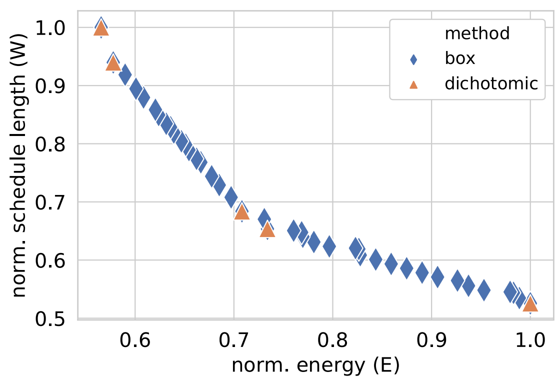


Fig. 8. Example scatterplot for set of Pareto-efficient solutions for Stencil, 10 tasks, CCR 0.1, 4 processors, linear speed, always everywhere.

For practical purposes, we set a high time limit (timeout) of three hours for the box method for each problem. For the dichotomic method, which only computes *supported* solutions, fewer solutions are expected. In an attempt to compute almost all of them, we use a still higher timeout of four hours. It is worth noting that here the term "timeout" refers to a limit on the time spent searching for the *set* of Pareto-efficient schedules for each problem, not a time limit applied to a single MIP solve. Also, the imposed time limits are not strict upper bounds. The Gurobi solver terminates at some point in time after the specified timeout limit. In some cases, it ran for an hour longer than the specified time limit.

6.2. Parameters

6.2.1. Graph inputs

In order to achieve our evaluation objectives we need a diverse set of task graphs which cover a broad spectrum of graph characteristics. At the same time – as we know that even optimal scheduling with only a single objective can take a long time – the graphs cannot be large or the number of graphs numerous.

With this in mind we selected four graph structures that are fundamental and representative: **fork-join**, **random**, **series-parallel**, **stencil**, illustrated in Fig. 7. We generated these graph structures in two different sizes, with **10 tasks** and **16 tasks**. For each graph, we randomly generated task and edge weights in such a way that we created three different computation to communication (CCR) ratios: **0.1 CCR**, **1 CCR**, **10 CCR**. The CCR is computed as the ratio of the sum of task work (node weights) to the sum of communication times (edge weights).

6.2.2. Parallel systems

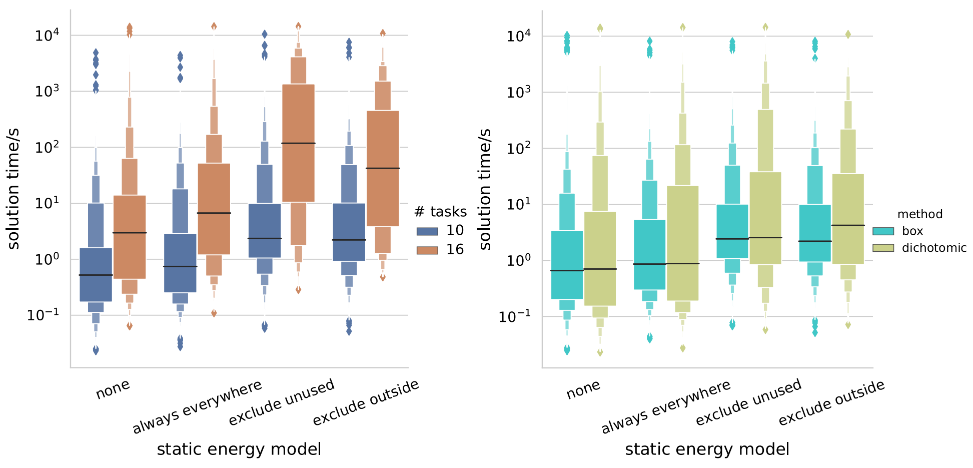
For each graph and method, the Pareto efficient schedules were computed for **two**, **four** and **six processors**. Two different speed sets were tested for each number of processors. A **linearly spaced set** $\mathcal{S} = \{0.8, 1.6, 2.4, 3.2\}$ and a **high-speed dominated set** $\mathcal{S} = \{0.5, 2.5, 3, 4\}$ were used for the ten-task models; In the sixteen-task models, the highest speeds were removed from each set in an attempt to reduce solution times. Static power consumption was set at one-tenth the maximum dynamic power with a corresponding scaling factor in the energy calculation.

6.2.3. Summary

In summary, 24 different graphs are scheduled (4 structures × 2 sizes × 3 CCR values) on six different target architectures (3 numbers of processors × 2 speed models). Four different MIP formulations are used, employing both the dichotomic and box methods to find sets of Pareto-efficient schedules. In total this leads to 1152 runs, each of which computes a set of Pareto-efficient schedules. Approximately one hundred computer days of solve time were required to search for solution sets for each problem. During that time more than 32,000 schedules were computed. More details are given next in the overview of Section 7.

7. Results

Each schedule that is produced for a given problem (input graph + number of processors + speed model + static energy (MIP) model) has a schedule length W and an energy consumption E. A set of Pareto-efficient schedules for the same problem usually consists of many



(a) Tasks (b) Methods

Fig. 9. Boxplot of solution time over static energy models.

different schedules (with different trade-offs between W and E). To be able to compare results across common parameters, we **normalise** the schedule length and the energy consumption to the respective maximum values for the set of solutions for the same parameters and thereby obtain values between 0 and 1.

As a first example, Fig. 8 shows a scatterplot over the normalised schedule length and the energy consumption of the schedules of the Pareto-efficient set found for the problem: Stencil, 10 tasks, CCR 0.1, 4 processors, linear speed, always everywhere static energy model. We distinguish between points obtained with the box method and the dichotomic method. When, as in this case, all non-dominated solutions have been computed, the solutions found by the dichotomic method are a subset (consisting of supported points) of the solutions found by the box method. This can be seen by the overlapping makers in the figure.

It is expected, and confirmed by our experimental results, that the number of supported points is significantly lower than the number of all non-dominated points. For this problem we computed five solutions with the dichotomic method and 46 with the box method. At the same time we observe that even for a small graph on four processors there is a good number of trade-offs between best schedule length and lowest energy consumption.

7.1. Overview

Before we further analyse the various aspects of the results, let us start with an overview of the obtained results. Table 2 gives an overview of the computed sets of Pareto-efficient schedules for the experimental workload. We distinguish the number of tasks and the graph structures and show the results for the two optimisation methods. The values shown are the number of times a complete set of Pareto-efficient schedules has been computed (and not timed out), how many times at least one schedule was found and the total number of schedules computed. For 10 tasks we are able to compute a complete set of Pareto-efficient schedules for the overwhelming majority of the scheduling problems. For graphs with 16 tasks, which are computationally more demanding, this is not the case. Having said that, in most cases, both for 10 and 16 tasks, we obtained at least one Pareto-efficient solution, with the notable exception of the fork-join graph structure with 16 tasks. Both methods struggle to find solutions for this graph type and size. This is not surprising, as it is known that fork-join graphs are difficult to schedule optimally, given the high degree of freedom from the lack of dependencies between most tasks (Orr and Sinnen, 2020).

As expected, the number of Pareto-efficient schedules (or corresponding non-dominated points) for a problem varies strongly between the two methods. This is simply due to the fact that the dichotomic method only finds supported points, whereas the box method can find all types of non-dominated points (i.e. supported and unsupported). For our input set the difference amounts to an order of magnitude in the total number of computed schedules.

Another expected observation that can be made based on the time-outs incurred is that the optimal task scheduling problem remains hard e.g. Davidović and Crainic, 2015; Mallach, 2018 and considering a bi-objective version of the task scheduling optimisation problem makes it even more challenging.

7.2. Static energy models

7.2.1. Solution time

Given the last observation, let us look at the behaviour under the different static energy models (Section 4.4). Fig. 9 depicts box plots of the **solution time** (in logarithmic scale) under the different static energy models.

It is important to note that the term "solution time" refers to the length of time taken to find *one* particular efficient schedule (or non-dominated point) of a particular problem instance and model. It should not be confused with the time taken to find a complete *set* of Pareto-efficient schedules. The use of the latter definition for a response variable could be problematic, as this duration is significantly dependent on the number of solutions.

It can be clearly seen that the static energy model has a significant impact on the difficulty of finding a Pareto-efficient schedule. The more elaborate models, i.e. the two *exclude* models, take longer to find a Pareto-efficient schedule, than the simpler models *none* and *always everywhere*. This is explained well by the more difficult MIP formulations for the more elaborate static energy models (Section 4.4). The difference is further pronounced with the higher number of 16 tasks (Fig. 9(left)), but remember also that within the time limit we found many more schedules for 10 tasks than for the 16 tasks problems. When distinguishing by the solution method, (Fig. 9(right)) there is very little difference for the average solution time (except under the *exclude outside* model), but the variance of the dichotomic method seems to be higher.

7.2.2. Shape of Pareto-efficient sets

Now that we have looked at the difficulty of MIP formulations for the different static energy models, let us look what the impact is on the nature of the obtained Pareto-efficient solutions.

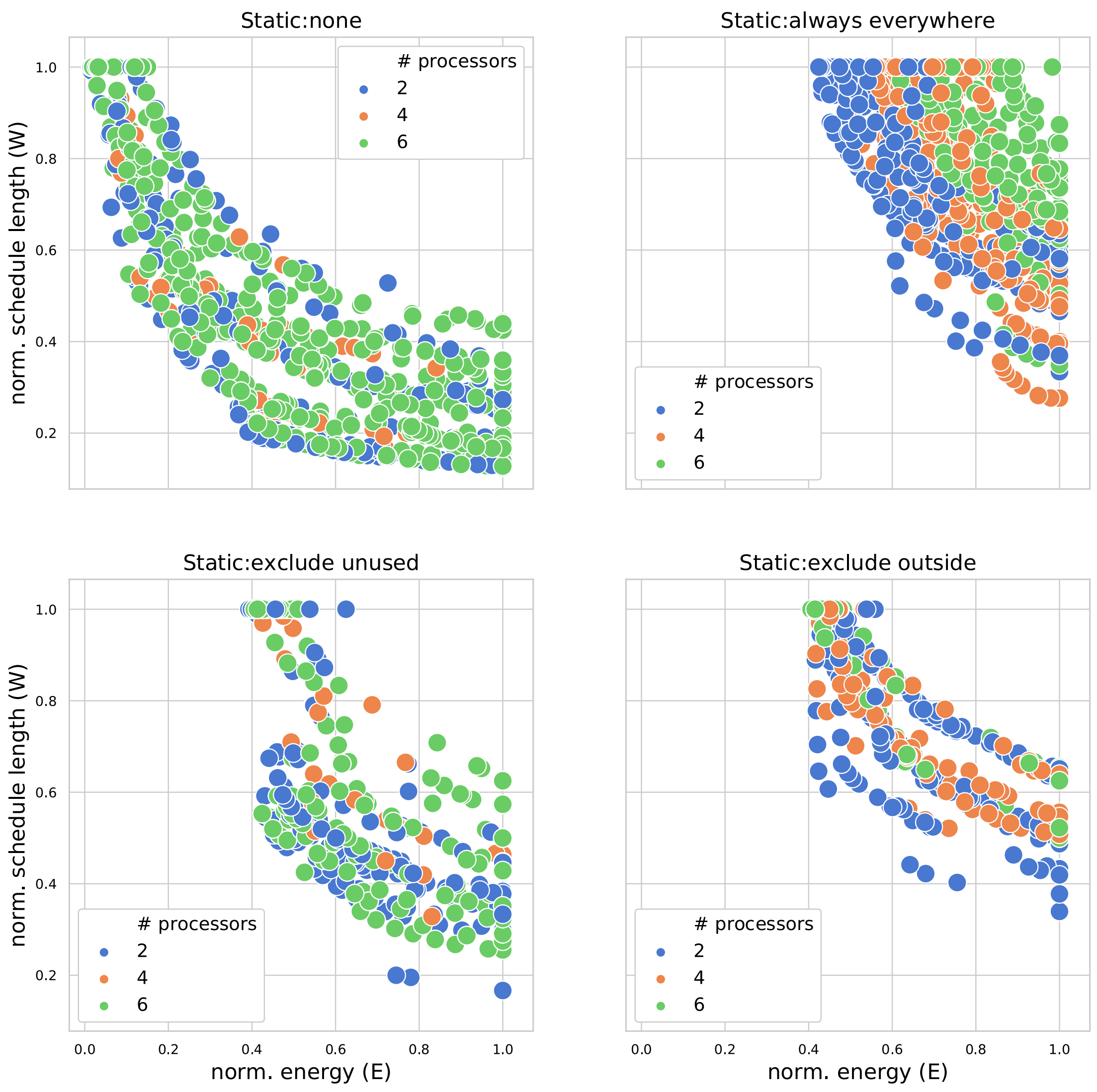


Fig. 10. Scatterplots for static energy models across all complete Pareto-efficient sets found with **dichotomic** method.

Table 2
Overview of the computed sets of Pareto-efficient schedules.

		Pareto-efficient set					
		Complete (max 72)		At least 1 (max 72)		# schedules	
	# tasks	10	16	10	16	10	16
Method	Structure						
box	**fork-join**	28	0	72	16	5075	20
	random	64	22	72	72	6348	1822
	series-parallel	64	3	72	72	6281	630
	stencil	68	11	72	72	7829	1233
dichotomic	**fork-join**	51	0	72	16	625	20
	random	68	29	72	72	688	338
	series-parallel	67	4	72	72	630	150
	stencil	69	27	72	72	505	308

Figs. 10 and 11 display four scatterplots, one for each static energy model, over all results with complete Pareto-efficient sets (no timeouts). Fig. 10 displays the results for the dichotomic method and Fig. 11 for the box method. The following observations apply to both methods. Data point colours are varied according to the number of processors. Thanks to the normalisation of the schedule length and the energy for each set we can make interesting comparisons here.

It is immediately clear that the shape of the scatter areas varies. When static energy in not considered (*none*, top-left), the typical half-moon shape of the scatter area is visible. For all other models, the scatter areas are much more compressed, especially in the energy dimension. This makes sense as with the consideration of static energy longer schedules (using processors at lower frequency) are not necessarily more energy efficient. In other words the non-dominated

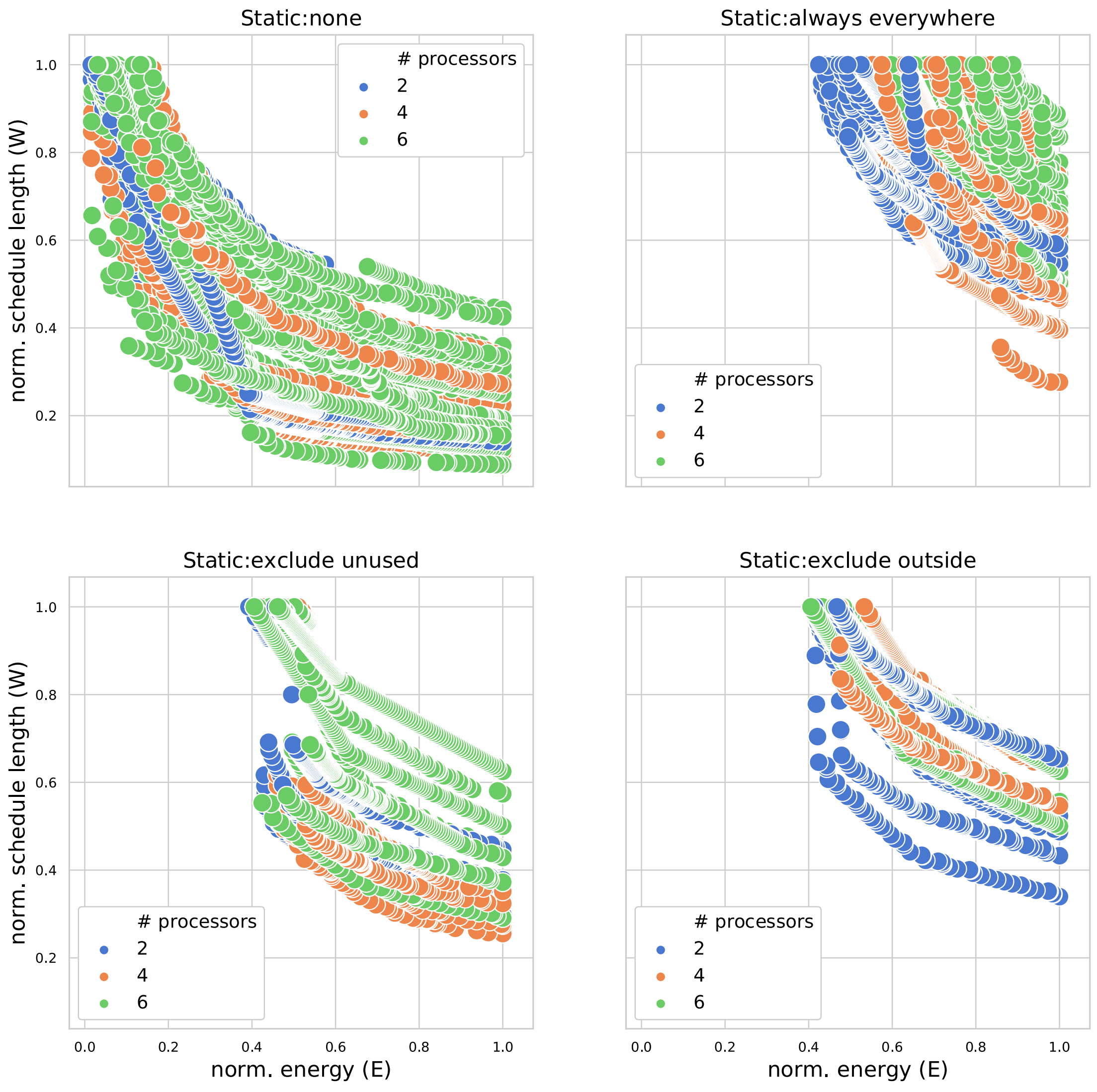


Fig. 11. Scatterplots for static energy models across all complete Pareto-efficient sets found with **box** method.

trade-off differences become smaller. A very important conclusion is that not considering static energy leads to the illusion that slowing down processors can save more energy than is actually realistic.

Under *always everywhere*(top-right) we see that the cluster area is smaller in general and seems to correlate with the number of processors. More processors reduce the variance in the results, which makes sense as even empty processors consume static energy and improvements in the schedule are limited by the static energy consumed on all processors until the schedule finishes.

The two *exclude* models (bottom-left and bottom-right) do not show this strong processor dependent behaviour, because empty processors do not consume energy, so moving tasks away from a processor to make it empty has a significant impact. This is most observable in the *exclude unused* model where the cluster area seems to be more spread out, likely due to the effect of including or excluding one or more processors. The *exclude outside* model comes closest to the half-moon shaped clustering and is arguable the most realistic model, which is a reaffirming outcome.

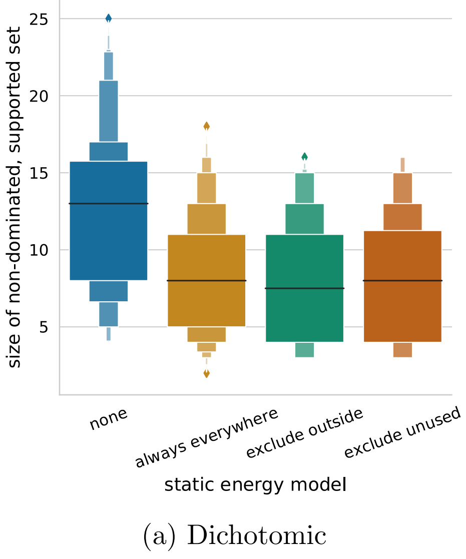
7.2.3. *Size of solution sets*

After looking at the distribution of the solutions for the different static energy models, we now look at the size of a complete set of Pareto-efficient solutions found for a problem. As we know, the sets found with the dichotomic method (supported efficient only) are significantly smaller than the sets found with the box method (all efficient). This is clearly visible in the two boxplots of Fig. 12, for all complete solution sets for 10 tasks.

More interesting to observe is that the size of the set depends on the static energy model. Without considering the static energy, *none*, there are significantly more solutions in the non-dominated sets. For the supported solution sets (Fig. 12(left)), there is very little difference between the other static models. Somewhat surprising is that, for all non-dominated solutions (Fig. 12(right)), the *always everywhere* model has statistically a considerably smaller set size than the two *exclude* models. This is possibly related to the smaller range between maximum to minimum energy consumption in the sets of the *always everywhere* model as can be observed in Fig. 11(top-right), most pronounced for four and six processors.

7.3. *Speed sets*

For our bi-objective task scheduling optimisation problems it is further interesting to study the impact of the two different speed sets, *linear* and *top heavy*.



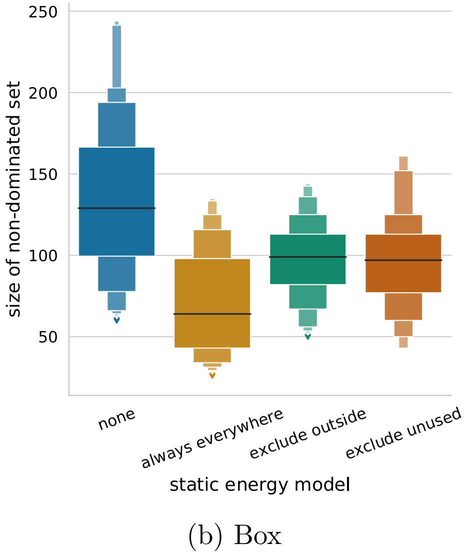


Fig. 12. Size of complete Pareto-efficient/non-dominated sets for 10 tasks.

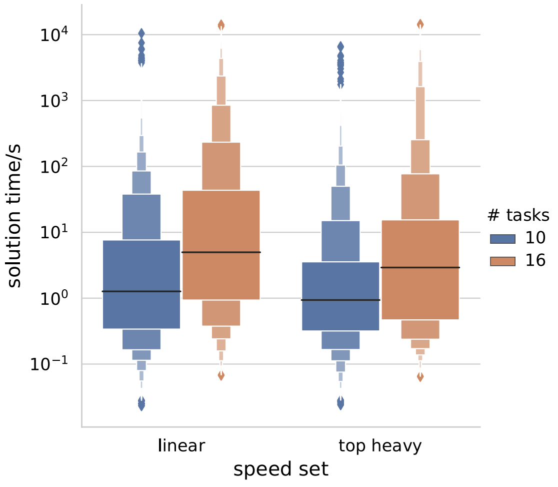


Fig. 13. Boxplot of solution time over processor speed sets.

7.3.1. *Solution time*

Let us first look if there is an impact on the difficulty of the problem, that is the solution time. The boxplots of the solution time in Fig. 13 show that there does not seem to be any discernible difference between the two speed sets, both *linear* and *top heavy* behave very similarly. This is not a great surprise as task processing times also do not seem to have any clear impact on the difficulty of the optimisation problem (Orr and Sinnen, 2020, 2021).

7.3.2. *Shape of Pareto-efficient sets*

The shape of the Pareto-efficient sets, however, is influenced by the processor speed sets. Fig. 14 depicts the scatterplots across the complete Pareto-efficient sets for static model *always everywhere* found with *dichotomic* method. For better visibility, we only depict the dichotomic method as the behaviour for both methods is similar (see Figs. 10 and 11).

The relatively even distribution of solution points we see for *linear* is more patchy for *top heavy*. This can be explained by the fact that the top heavy speed set has less lower speeds and might not be able to achieve the desired processor speed and thereby trade-off between schedule length and energy.

A number of the graph characteristics tested did not have an obvious effect on solution times. These included number of processors, speed set type, and computation to communication ratio. It should be stressed that these covariates could affect solve times, but a more rigorous statistical study, performed on a larger dataset would be necessary to draw valid conclusions here.

8. Conclusion

Addressing an important research gap, this paper proposed the first bi-objective MIP for the task scheduling problem with communication delays. In contrast to previous work, it addresses two important aspects of parallel computing simultaneously and in a Pareto-efficient manner: fast application execution and efficient energy usage. This MIP was varied to incorporate static energy consumption to create additional, realistic model variants.

The different bi-objective formulations were then used to perform the first extensive experimental study of sets of Pareto-efficient solutions for the task scheduling problem, using a large set of test problems and employing two different bi-objective solution methods.

This experimental campaign revealed a connection between the static energy model and the size and distribution of the set of non-dominated Pareto-efficient solutions. It demonstrated that the dichotomic method is practical to find few, but supported non-dominated solutions, whereas the box method can find a complete set of non-dominated solutions, but is computationally very expensive given the substantially larger number of solutions to be found. The static energy model has a significant impact on the shape of the Pareto-efficient sets and showed that the consideration of static energy is important to make the right trade-off between schedule length and energy consumption. These insights can have practical implications on how scheduling heuristics should behave in practice.

Future work will need to investigate varied and improved MIP formulations that can be solved in significantly less time. Motivated by the evolution of optimal scheduling algorithms for a single objective this seems to be a possibility. Having better MIP formulations will allow to solve larger problems in a reasonable time to gain further insights into the characteristics and shapes of sets of Pareto-efficient solutions for this important and classical scheduling problem.

Another interesting avenue for future research might be a smart combination of the two solution methods, dichotomic method and the box method, to find more solutions quicker.

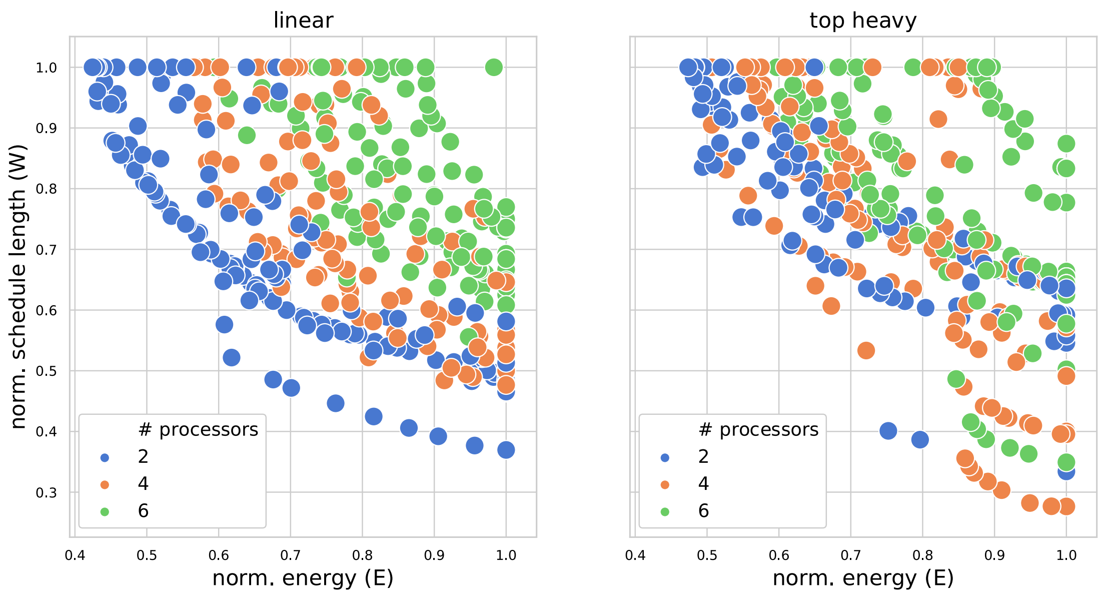


Fig. 14. Scatterplots for processor speed sets across the complete Pareto-efficient sets for static model *always everywhere* found with *dichotomic* method.

CRediT authorship contribution statement

Russell Stewart: Methodology, Software, Formal analysis, Investigation, Data curation, Writing – original draft, Visualization. **Andrea Raith:** Conceptualization, Methodology, Validation, Formal analysis, Resources, Writing – review & editing, Visualization, Supervision. **Oliver Sinnen:** Conceptualization, Methodology, Validation, Formal analysis, Resources, Data curation, Writing – review & editing, Visualization, Supervision.

Data availability

Data will be made available on request.

References

Ahmad, I., Ranka, S., Khan, S.U., 2008. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In: 2008 IEEE International Symposium on Parallel and Distributed Processing. IEEE, pp. 1–6.

Aneja, Y.P, Nair, K.P.K., 1979. Bicriteria transportation problem. Manage. Sci. 25, 73–78.

Anghinolfi, D., Paolucci, M., Ronco, R., 2021. A bi-objective heuristic approach for green identical parallel machine scheduling. European J. Oper. Res. 289 (2), 416–434.

Arora, M., Manne, S., Paul, I., Jayasena, N., Tullsen, D.M., 2015. Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on CPU-GPU integrated systems. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture. HPCA, IEEE, pp. 366–377.

Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.-A., 2011. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. Concurr. Comput. : Pract. Exp. 23 (2), 187–198.

Aupy, G., Benoit, A., Dufossé, F., Robert, Y., 2013. Reclaiming the energy of a schedule: Models and algorithms. Concurr. Comput. : Pract. Exp. (ISSN: 15320626) 25 (11), 1505–1523.

Benoit, A., Melhem, R., Renaud-Goud, P., Robert, Y., 2013. Assessing the performance of energy-aware mappings. Parallel Process. Lett. 23 (2), 1340003.

Boland, N., Charkhgard, H., Savelsbergh, M., 2017. The quadrant shrinking method: A simple and efficient algorithm for solving tri-objective integer programs. European J. Oper. Res. 260 (3), 873–885.

Chankong, V., Haimes, Y., 1983. Multiobjective Decision Making: Theory and Methodology. Elsevier Science Publishing Co., New York.

Chen, J., He, Y., Zhang, Y., Han, P., Du, C., 2022. Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems. J. Syst. Archit. 129, 102598.

Cohon, J.L., Church, R.L., Sheer, P., 1979. Generating multiobjective trade-off: An algorithm for bicriterion problems. Water Resour. Manag. 15, 1001–1010.

Cojean, T., Guermouche, A., Hugo, A., Namyst, R., Wacrenier, P.-A., 2019. Resource aggregation for task-based Cholesky factorization on top of modern architectures. Parallel Comput. 83, 73–92.

Colin, A., Kandhalu, A., Rajkumar, R.R., 2016. Energy-efficient allocation of real-time applications onto single-ISA heterogeneous multi-core processors. J. Signal Process. Syst. 84 (1), 91–110.

Davidović, T., Crainic, T.G., 2015. Parallel local search to schedule communicating tasks on identical processors. Parallel Comput. 48, 1–14.

Davidović, T., Liberti, L., Maculan, N., Mladenović, N., 2007. Towards the optimal solution of the multiprocessor scheduling problem with communication delays. In: Proc. 3rd Multidisciplinary Int. Conf. on Scheduling: Theory and Application. MISTA, pp. 128–135.

Dietze, R., Rünger, G., 2020. The search-based scheduling algorithm HP* for parallel tasks on heterogeneous platforms. Concurr. Comput.: Pract. Exper. e5898.

Doğan, S.F., Karsu, Ö., Ulus, F., 2021. An exact algorithm for biobjective integer programming problems. Comput. Oper. Res. 132, 105298.

Drozdowski, M., 2009. Scheduling for Parallel Processing. Springer.

Ehrgott, M., 2005. Multicriteria Optimization, second ed. Springer.

Ehrgott, M., Ryan, D., 2002. Constructing robust crew schedules with bicriteria optimization. J. Multi-Criteria Decis. Anal. 11, 139–150.

Eichfelder, G., 2008. Adaptive Scalarization Methods in Multiobjective Optimization. Springer.

Eitschberger, P., Keller, J., 2020. Comparing optimal and heuristic taskgraph scheduling on parallel machines with frequency scaling. Concurr. Comput.: Pract. Exper. 32 (10).

Fang, K., Uhan, N.A., Zhao, F., Sutherland, J.W., 2013. Flow shop scheduling with peak power consumption constraints. Ann. Oper. Res. 206, 115–145.

Gadegaard, S.L., Nielsen, L.R., Ehrgott, M., 2019. Bi-objective branch-and-cut algorithms based on LP relaxation and bound sets. INFORMS J. Comput. 31 (4), 790–804.

Grama, A., Gupta, A., Karypis, G., Kumar, V., 2003. Introduction to Parallel Computing, second ed. Pearson, Addison Wesley, UK.

Halffmann, P., Schäfer, L.E., Dächert, K., Klamroth, K., Ruzika, S., 2022. Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey. J. Multi-Criteria Decis. Anal. 29 (5–6), 341–363.

Hamacher, H.W., Pedersen, C.R., Ruzika, S., 2007. Finding representative systems for discrete bicriterion optimization problems. Oper. Res. Lett. 35, 336–344.

Hu, B., Cao, Z., Zhou, M., 2019. Scheduling real-time parallel applications in cloud to minimize energy consumption. IEEE Trans. Cloud Comput..

Huang, K., Jiang, X., Zhang, X., Yan, R., Wang, K., Xiong, D., Yan, X., 2018. Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems. IEEE Access 6, 57614–57630.

Lee, Y.C., Zomaya, A.Y., 2010. Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans. Parallel Distrib. Syst. 22 (8), 1374–1381.

Li, Z., Yang, H., Zhang, S., Liu, G., 2016. Unrelated parallel machine scheduling problem with energy and tardiness cost. Int. J. Adv. Manuf. Technol. 84, 213–226.

Liu, J., Yang, P., Chen, C., 2023. Intelligent energy-efficient scheduling with ant colony techniques for heterogeneous edge computing. J. Parallel Distrib. Comput. 172, 84–96.

Malik, A., Walker, C., O´Sullivan, M., Sinnen, O., 2017/2018. Satisfiability modulo theory (SMT) formulation for optimal scheduling of task graphs with communication delay. J. Comput. Oper. Res. 89C, 113–126.

Mallach, S., 2018. Improved mixed-integer programming models for the multiprocessor scheduling problem with communication delays. J. Comb. Optim. 36, 871–895.

Olukotun, K., Hammond, L., 2005. The future of microprocessors. Queue (ISSN: 1542-7730) 3 (7), 26–29.

Orr, M., Sinnen, O., 2020. Optimal task scheduling benefits from a duplicate-free state-space. J. Parallel Distrib. Comput. 146, 158–174.

Orr, M., Sinnen, O., 2021. Optimal task scheduling for partially heterogeneous systems. Parallel Comput. 107.

Parragh, S.N., Tricoire, F., 2019. Branch-and-bound for bi-objective integer programming. INFORMS J. Comput. 805–822.

Pillai, A.S., Singh, K., Saravanan, V., Anpalagan, A., Woungang, I., Barolli, L., 2018. A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems. Soft Comput. 22 (10), 3271–3285.

Pruhs, K., van Stee, R., Uthaisombut, P., 2008. Speed scaling of tasks with precedence constraints. Theory Comput. Syst. 43, 67–80.

Qin, Y., Zeng, G., Kurachi, R., Li, Y., Matsubara, Y., Takada, H., 2019. Energy-efficient intra-task DVFS scheduling using linear programming formulation. IEEE Access 7, 30536–30547.

Quan, Z., Wang, Z.-J., Ye, T., Guo, S., 2019. Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems. IEEE Trans. Parallel Distrib. Syst. 31 (5), 1165–1182.

Rayward-Smith, V.J., 1987. UET scheduling with unit interprocessor communication delays. Discrete Appl. Math. 18, 55–71.

Roy, S.K., Devaraj, R., Sarkar, A., 2019a. Optimal scheduling of PTGs with multiple service levels on heterogeneous distributed systems. In: 2019 American Control Conference. ACC, pp. 157–162.

Roy, S.K., Devaraj, R., Sarkar, A., Maji, K., Sinha, S., 2020. Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems. J. Syst. Archit. 105, 101706.

Roy, S.K., Devaraj, R., Sarkar, A., Sinha, S., Maji, K., 2019b. Optimal scheduling of precedence-constrained task graphs on heterogeneous distributed systems with shared buses. In: 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing. ISORC, pp. 185–192.

Safari, M., Khorsand, R., 2018. Energy-aware scheduling algorithm for time-constrained workflow tasks in DVFS-enabled cloud environment. Simul. Model. Pract. Theory 87, 311–326.

Sinnen, O., 2007. Task Scheduling for Parallel Systems. Wiley.

Sinnen, O., 2014. Reducing the solution space of optimal task scheduling. Comput. Oper. Res. 43, 201–214.

Stidsen, T., Andersen, K.A., Dammann, B., 2014. A branch and bound algorithm for a class of biobjective mixed integer programs. Manage. Sci..

Tang, Q., Wu, S., Shi, J., Wei, J., 2017. Optimization of duplication-based schedules on network-on-chip based multi-processor system-on-chips. IEEE Trans. Parallel Distrib. Syst. 28 (3), 826–837.

Tang, Q., Zhu, L.H., Zhou, L., Xiong, J., Wei, J.B., 2020. Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems. J. Parallel Distrib. Comput. 138, 115–127.

Venugopalan, S., Sinnen, O., 2015. ILP formulations for optimal task scheduling with communication delays on parallel systems. IEEE Trans. Parallel Distrib. Syst. (ISSN: 1045-9219) 26 (1), 142–151.

Venugopalan, S., Sinnen, O., 2016. Memory limited algorithms for optimal task scheduling on parallel systems. J. Parallel Distrib. Comput. 92, 35–49.

Wang, L., von Laszewski, G., Dayal, J., Wang, F., 2010. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In: Proceedings of the IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. CCGRID, pp. 368–377.

Wang, S., Wang, X., Yu, J., Ma, S., Liu, M., 2018. Bi-objective identical parallel machine scheduling to minimize total energy consumption and makespan. J. Clean. Prod. 193, 424–440.

Wu, X., Che, A., 2019. A memetic differential evolution algorithm for energy-efficient parallel machine scheduling. Omega 82, 155–165.

Xie, G., Huang, J., Li, Y.L.R., Li, K., 2019. System-level energy-aware design methodology towards end-to-end response time optimization. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst..

Xie, G., Xiao, X., Peng, H., Li, R., Li, K., 2022. A survey of low-energy parallel scheduling algorithms. IEEE Trans. Sustain. Comput. 7 (1), 27–46.

Zhang, Y., Wang, Y., Tang, X., Yuan, X., Xu, Y., 2018. Energy-efficient task scheduling on heterogeneous computing systems by linear programming. Concurr. Comput.: Pract. Exper. 30 (19), e4731.

Zhou, J., Sun, J., Cong, P., Liu, Z., Zhou, X., Wei, T., Hu, S., 2019. Security-critical energy-aware task scheduling for heterogeneous real-time MPSoCs in IoT. IEEE Trans. Serv. Comput. 13 (4), 745–758.

Zhu, D., Melhem, R., Mossé, D., 2004. The effects of energy management on reliability in real-time embedded systems. In: IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004, IEEE, pp. 35–40.