2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 979-8-3503-6460-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/IPDPSW63119.2024.00166

State-Space Search to Find Energy-Aware Pareto-Efficient Optimal Task Schedules

Yasith Udagedara*, Andrea Raith†, Oliver Sinnen*
University of Auckland, New Zealand
Email: yuda947@aucklanduni.ac.nz, a.raith, o.sinnen@auckland.ac.nz
* Department of Electrical, Computer, and Software Engineering † Department of Engineering Science

***Abstract*—Bi-objective scheduling for parallel computing, in particular with the objectives to minimize makespan and energy, has been well studied in the literature. Numerous works have proposed algorithms that use Dynamic Voltage and Frequency Scaling (DVFS) as the mechanism to reduce energy consumption. Given the NP-hard nature of the combinatorial task scheduling problem, these algorithms are mostly heuristics, producing only a single or few schedules for a given instance.**

This research explores bi-objective task scheduling with the goal to find the set of Pareto-efficient solutions for a given small instance, using DVFS to balance energy and performance. We undertake this using a branch-and-bound depth-first search within an allocation-ordering state space model. Varying scaling factors and static energy models are considered. The proposed approach is evaluated with a dataset of small graphs resulting in thousands of produced schedules. In contrast to heuristics, our approach computes numerous trade-off solutions for a given problem instance as a Pareto-efficient set.

I. Introduction

As the computational requirements of modern applications continue to rise, the performance of single computational units has failed to keep up [1]. Parallel processing is one of the predominant concepts used in this regard [2]. In this research, we focus on one of the critical parts of any parallel processing task, which is task scheduling. It involves organizing a given set of tasks onto a set of processors, in space and time, while adhering to the dependencies. The time taken to complete all tasks in a given computational load is called the schedule length or makespan. The typical goal of task scheduling is to reduce the makespan. Unfortunately, this is a well-known NP-hard problem [3], meaning that it cannot be solved in polynomial time of the number of tasks. As typical of NP-hard problems, many task scheduling heuristics have been proposed [4], [5]. However, certain situations benefit from optimal schedules [6], [7], for example when a program (and hence its schedule) is executed many times. Optimal schedules are also important to inform designers of heuristics as a lower bound and natural benchmark. It is self-evident that energy consumption has become an important consideration in parallel computing and hence task scheduling [8]. A standard technique to reduce the power-consumption and in turn the energy consumption of a schedule is to use Dynamic Voltage and Frequency Scaling (DVFS) of processors. In this technique, the clock frequency of a processor (or core) is lowered, resulting in a slower execution, but also a lower energy consumption. Finding fast schedules that minimize the energy consumtion for the execution of a program is an important goal, especially in the form of trade-offs that satisfy certain constraints or limits on either of the objectives. Bi-criterion task scheduling considers two criteria at the same time [9], mostly employing heuristic-based methods. Bi-criterion optimal task scheduling is mostly unexplored [10].

This paper proposes a task scheduling technique for parallel computing based on a new state space search-based approach for optimal, i.e. Pareto-efficient bi-criterion task scheduling. We target the scheduling of task graphs with communication costs on a set of homogeneous processors, which is a classic scheduling problem. The scheduling approach is based on a depth-first branch-and-bound algorithm that identifies the complete set of Pareto-efficient schedules for a given problem instance. In this research we focus on small graphs and a set of processors whose execution speed can be adjusted.

We explore a variety of different factors that affect bi-criterion task scheduling and perform thorough experiments to measure the performance of our solution method and gather insights into bi-criterion task scheduling.

The major contributions of this paper are:

- Adjusting the task scheduling state-space model for bi-criteria optimisation and processor speed-scaling
- Proposing data-structures and bounding techniques for the bi-criteria search
- A simulation-based evaluation of the proposed approach with a large set of task graphs and varying number of processors.

The rest of this paper is organized as follows. Section II goes through the related work. Section III formally defines the bi-criterion task scheduling problem and the aims of our approach. Section IV explains the proposed state space-based solution. Section V highlights the experimentation methodology. Section VI runs through the results of the experiments and our discussion on them. Concluding remarks are made in Section VII.

II. Related work

A. Heuristic Task scheduling

As task scheduling is a NP-hard problem [11], heuristics are commonplace solution methods [4], [5], [12]. Even though they do not produce optimal solutions, they are the only

979-8-3503-6460-6/24/$31.00 ©2024 IEEE
DOI 10.1109/IPDPSW63119.2024.00166

practically feasible option when dealing with a larger number of tasks. Many such heuristics have been proposed [4], [5], and there have also been attempts to compare popular heuristics to measure their performance [12].

B. Single criterion optimal task scheduling

Even though heuristics are sufficient for most scenarios, optimal schedules are crucial in certain situations. Several methods have been adapted to solve this problem from a variety of combinatorial optimization fields. These include state space search and Multiple Integer Linear Programming (MILP).

State-space search [13], [14], [15] , breaks down each possible schedule and partial schedule into a set of states and connections among them. A state space can be thought of as a tree graph. A traversal algorithm such as branch-and-bound or A-star is used in this spate-space to find the best possible solution. The type of state space used in state space search is an important consideration. Different state space models have their own upsides and drawbacks. The most common state space draws inspiration from list scheduling [16] and has all possible list schedules for the subsets of tasks as the states. Although this state space is popular, it has many drawbacks, the chief of which is duplicate states. The Allocation-Ordering (AO) state space [15] was created with the intention of eliminating duplicate states.

Mathematical programming has been applied by [17], [18], [19], [20], [21], [22], [23], [24] to optimally solve single-criterion scheduling problems. Most of these articles propose MILP formulations, with the exception of [25], where Satisfiability Modulo Theory (SMT) is used.

C. Task scheduling with energy-awareness

Often makespan, or in general execution performance in other scheduling domains, is not the only optimization criterion. Sometimes more than one criterion is optimized, such as energy [26] or fault tolerance [27]. We focus here on energy as the second criterion.

In constrained-based algorithms, we have two different approaches: minimizing makespan under an energy consumption threshold [28] and minimizing energy consumption under a makespan threshold [29].

A few MILP-based algorithms have been proposed for the two approaches, [30], [31], [32], [33], [34], which are mostly in the area of scheduling with real-time constraints [31], [32], [33], [34]. [30], [31], [32], [33] are optimising energy under performance constraints while [34] optimises performance under energy constraints.

D. Bi-criteria task scheduling

Bi-criterion task scheduling involves optimizing both criteria at the same time, as opposed to constrained-based task scheduling, which only optimises one criterion but the constraint of another. There is not a lot of research done on this particular subject, a few example are [35], [36], [37] – often in the real-time scheduling area, and the techniques used are widely different. These techniques include balanced task duplication [38] and regression learning [39].

However, all of these techniques are not focused on optimal scheduling. Research on optimal bi-criterion scheduling is rare. In MILP-based approach, the authors of [10] compute Pareto-efficient sets of solutions for small task graphs. In this work we have the same goal, but with explore a state-space search instead of a MILP-based solution.

III. Task scheduling problem

Our goal is to schedule a set of tasks onto a set of processors. Each task carries a weight to it, which is a representation of its computational time cost. In addition, some tasks can be started only when other tasks have finished, using the output of the other tasks as their input, which are hereby referred to as precedence constraints. The corresponding communication of data incurs a communication cost if the two incident tasks are processed on different processors.

Formally, the task scheduling problem is represented using a directed acyclic graph (DAG) $G = \{V, E, W, C\}$, known as task graph. V represents the set of tasks $n_i \in V$, while E represents the edges (precedence constraints) $e_{ij} \in E$, where task n_i is required to finish before task n_j can be started. The weight of task $n_i \in V$ is $w_i \in W$, and $c_{ij} \in C$ is the communication cost associated with precedence constraint $e_{ij} \in E$ when the task n_i and n_j are executed on different processors. Otherwise the communication cost betweem tasks on the same processor is assume to be negligible and set to 0. It should be noted that all task weights and edge weights are integers.

Figure 1 shows a simple task graph. n_1, n_2, n_3, $n_4 \in V$ are tasks with weights w_1, w_2, w_3, $w_4 \in W$ respectively. Three precedence constraints are established, $e_{12}, e_{13}, e_{34} \in E$ with communication costs c_{12}, c_{13}, $c_{34} \in C$ respectively.

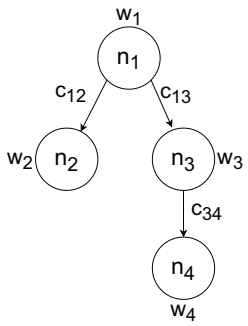


Fig. 1: A sample task graph

A. System model

The target system onto which the task graph is scheduled is assumed to consist of a set of processors P. The processors are homogeneous and their frequency can be changed instantly with no cost. The processor frequency influences the energy consumption and speed of execution of tasks. Higher

frequencies consume more power, but process tasks quicker. Therefore task duration will depend on linearly on frequency, determined by (1), where d_{ik} refers to the time that task n_i with weight w_i takes to complete under frequency $f_k \in F$. F is known as the speed set, the finite set of discrete frequencies the processor can operate under.

$$d_{ik} = \frac{w_i}{f_k} \tag{1}$$

While modern processors can be able to adjust processing speeds in a continuous or at least very fine grained fashion, we work here with a finite set of speeds to create a combinatorial optimisation problem. One task may only be run under a single frequency, i.e. the frequencies is not changed during its execution, and no task may be preempted from completing its execution. The target system has a communication subsystem which connects all processors equally, i.e. with equal communication times, and communication is assumed to happen without contention and without processor involvement.

B. Task schedule and makespan objective

A valid schedule S for a given task graph G and set of processors P is an allocation of the tasks to the processors at specific start times while satisfying precedence constraints ensuring no tasks within the same processor overlap. Within a schedule, each task n_i is assigned to a processor $assigned_i \in P$. Each task is scheduled to start at a particular time $start_i$ and will finish at $finish_i$ when its task duration d_{ik} has passed, i.e. $finish_i = start_i + d_{ik}$.

Precedence constraints ensure that tasks can only start after completion of their predecessor tasks (if any). Communication costs also need to be considered if the two tasks are scheduled in different processors. This is represented by (2).

$$\begin{aligned} &start_j \geq finish_i + z_{ij} \cdot c_{ij} \quad \forall e_{ij} \in E \\ &z_{ij} = \left\{ \begin{array}{ll} 1, & \text{if } assigned_i \neq assigned_i \\ 0, & \text{otherwise} \end{array} \right\} \end{aligned} \tag{2}$$

A non-overlapping condition ensures that no two tasks are scheduled in the same processor at the same time, as follows:

$$\begin{aligned} (assigned_i = assigned_j) \wedge (start_i \leq start_j) \\ \implies finish_i \leq start_j \qquad \forall n_i \in V \end{aligned} \tag{3}$$

A schedule has a makespan or schedule length $MS(S)$. This is the total time taken to finish all tasks. It is the time when the last task finishes, i.e. $MS(S) = max(finish_i)$.

A sample task schedule of 5 tasks (different to graph of Figure 1) onto 4 processors is shown in Figure 2. We can observe the precedence constraints from n_1 to n_2, n_3 and n_4 preventing them from being started earlier. n_2 starts immediately after n_1 as they are scheduled on the same processor. On the other hand, n_3 and n_4 have started after a delay owing to different communication costs. We can also notice that the makespan of the schedule is the finish time of task n_5, which is the last task to finish.

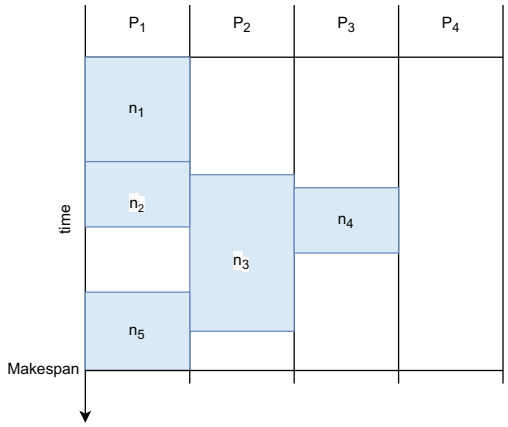


Fig. 2: A sample task schedule

C. Energy consumption in task schedules

We consider two types of energy in our research: static energy $static_{tot}$ and dynamic energy $dynamic_{tot}$, and their sum is the total energy consumption $energy_{tot}$.

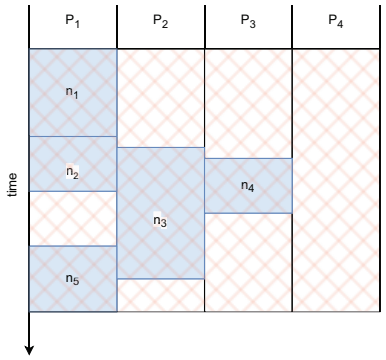
$$energy_{tot}(S) = dynamic_{tot}(S) + static_{tot}(S) \tag{4}$$

Dynamic energy $dynamic_{tot}$ is used when operating tasks and is modelled according to Dynamic Voltage and Frequency Scaling (DVFS) [40]. Commonly, the dynamic *power* consumption is described to have a cubic relation to the frequency f, i.e. $\propto f^3$. Following this, the dynamic energy consumption $dynamic_{ik}$ for a task of weight w_i running with frequency f_k for a duration d_{ik} is given in (5), where C_d is a constant and the duration is replaced with (1) in the last term.

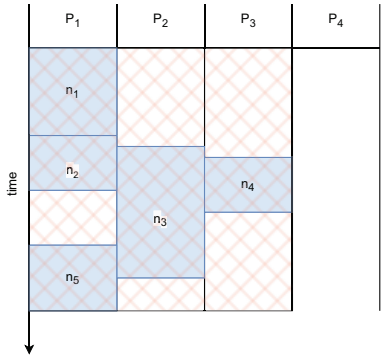
$$dynamic_{ik} = C_d \cdot f_k^3 \cdot d_{ik} = C_d \cdot w_i \cdot f_k^2 \tag{5}$$

D. Static energy

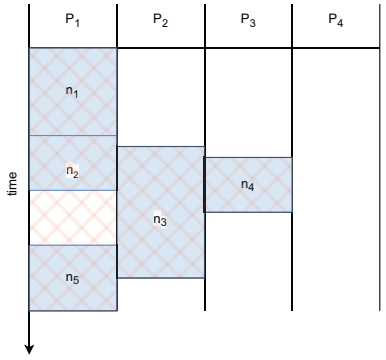
Static energy $static_{tot}$ is the constant energy consumption throughout the operation of the processor. The power consumed by static energy is referred to by ω. In order to have realistic considereing of static energy, we explore four different static energy models in this research: none, always everywhere (AE), exclude unused (EU), and exclude outside (EO). The 'none' model assumes zero static energy within the system. The AE model considers the entire schedule length across all processors of the target system. The EU model considers the entire schedule length across all active processors P_{active} (processors in which at least one task is scheduled). This model reflects that processors can be completely deactivated when not being part of the schedule (but part of the target system). The EO model considers the duration from the moment the first task scheduled on the processor is started until the last task is finished. With this model we further reflect that processors do not need to be active before the first tasks needs to be executed and after the the last task allocated to them has finished.



(a) Always everywhere



(b) Exclude unused



(c) Exclude outside

Fig. 3: Visualisations of the different static energy models

Figure 3 shows a sample schedule with different static energy models, with the cross-hatch pattern showing the period for which static energy is applicable. We will later see that the different static energy models have an influence on the shape of the Pareto-efficient sets, giving interesting insights.

E. Objective

The objective is to compute a set of Pareto-efficient task schedules for a given task graph and target system where the two objectives that are considered are makespan and energy cost. A particular schedule is called Pareto-efficient if there is no other schedule that dominates it, i.e. that improves at least one of the criteria without worsening the other. A schedule is said to be dominated by another schedule if makespan, energy consumption, or both are lower in the dominating solution compared to the dominated solution. Therefore a Pareto-efficient schedule is also called non-dominated. For each pair of schedules S_1, S_2 in the Pareto-efficient set PE, we therefore know that $MS(S_1) < MS(S_2)$ implies $energy_{tot}(S_2) < energy_{tot}(S_1)$.

IV. Proposed algorithm

With the given definition of the scheduling problem, this section details the proposed approach to compute the Pareto-efficient set.

A. State spaces

The technique used to address the problem is state space search. A state space is a discrete and distinct set of states that represents all possible configurations of a solution to a problem. In our case, the state space is a complete representation of all possible schedules, including partial schedules (a partial schedule is a schedule that has a subset of tasks scheduled but not all). Each state represents such a schedule. The states in a state space are connected by a set of operations. In this context, the operation is adding a task to a partial schedule. This means that two states are connected if their schedules only differ by one task (the organization of all other tasks in the two schedules should be identical). This creates a tree known as the state space. The leaf nodes of this tree provide all the possible complete schedules for the problem instance.

B. State space search

State space search is any algorithm that can be used to search through a state space. As state spaces are graphs, graph search algorithms are mostly used for this purpose. The most common search algorithms used in state space search are A-star and depth-first search (DFS). DFS is a simple algorithm, in which we start at the initial state and explore until we reach a complete solution. After that, we backtrack along the explored path and explore unexplored states. DFS is combined with branch-and-bound (BNB) to improve its performance. Branch-and-bound is the use of a bounding function to prune any parts of the state space that are not worth exploring. Bounding is explained in detail in Section IV-G. We use a DFS BNB algorithm in this research, as this has been shown to be superior to A-star, especially due to dramatically lower memory consumption [41].

C. State space model

In this research, we use the Allocation-Ordering(AO) state-space model [15]. The AO state space model divides the scheduling into two phases: allocation and ordering. The allocation state space is built with the goal of finding all possible processor allocations for all tasks. We start with an empty allocation, and at each child state, we add a new task into the allocation. For homogeneous processors, this is

similar to a set partitioning with a fixed number of maximum partitions. When all tasks have been allocated, the allocation phase is completed.

The ordering phase begins where the allocation ends. It takes in a full allocation of tasks to processors and orders them while maintaining the conditions required to create a valid schedule, as outlined in III-B. The ordering phase begins with an empty ordering, and at each child state, a new task is ordered. When all tasks have been ordered, we get the final valid schedule.

D. Speed factors

The different clock frequencies at which a processor runs determine the execution duration of a task according to (1). While the task weights $w_i \in W$ are integers, this would lead in general to non-integer durations. To maintain integer values for better comparability and accurate calculation of precedence constraint satisfaction, we use mathematically equivalent speed factors instead of frequencies in our work. Speed factors are integers and convert the task duration function into a multiplication operation; thus, ensuing task durations are integers. The set of speed factors SF are calculated as follows, where speed factor $sf_k \in SF$ corresponds to frequency f_k and $LCM(F)$ refers to the least common multiple of all frequencies in *F*.

$$sf_k = \frac{LCM(F)}{f_k} \tag{6}$$

Using speed factors, we can define the task duration d'_{ik} for task w_i with speed factor sf_k as follows. All w_i and sf_k are integers, therefore all d'_{ik} are integers as well.

$$d'_{ik} = w_i \cdot sf_k \tag{7}$$

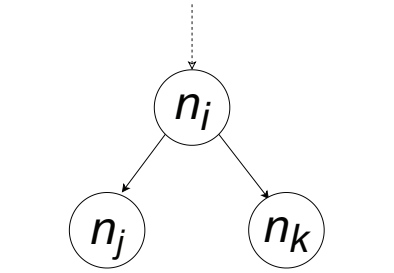
Using (5), the dynamic energy consumption $dynamic_{ik}$ for a task of weight w_i running with speed factor sf_k for a duration d'_{ik} is as follows.

$$dynamic_{ik} = \frac{w_i}{sf_k^2} \tag{8}$$

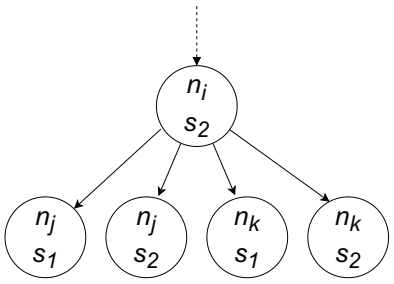
E. Speed scaling in state space search

While we adopt the AO state space model, it is oblivious to different execution speeds and simply assumes all tasks are executed at full speed. In order to integrate speed scaling into the AO model, we had the choice of doing this in the A (allocation) or O (ordering) phase. It seems meaningful to do this in the ordering phase because a given allocation provides more information about slack in the schedule, which can be exploited by speed scaling. Each ordering state not only orders a previously unordered task but also assigns a speed factor to it. This creates multiple states for the same partial ordering, where previously only one was made (one for each speed factor considered). This causes a higher branching factor in the ordering phase, resulting in a higher number of states to be considered, but the depth of the search space remains the same.

Figure 4 shows an example of the comparison between the original AO model and the modified AO model used in our research. $(n_i, n_j, n_k) \in V$ are the new nodes being ordered in the states, and $(s_1, s_2) \in S$ are the speed factors for the newly ordered nodes (e.g., $speed(n_i) = s_2$ in the parent node in Figure 4b). All other information about the nodes is abstracted away for the sake of clarity, and not all children of the parent node are shown. We can notice the increase in the number of states when using frequency scaling, which has an exponential effect as we travel further down the state space.



(a) Without frequency scaling



(b) With frequency scaling

Fig. 4: A section of the ordering state space with and without frequency scaling

F. Non-dominated solution set

In single criterion search, a DFS Branch and Bound algorithm records and updates the best schedule found, which is the optimal schedule when the search completes. In bi-criteria search, only recording one best solution is obviously not sufficient. Instead, it is necessary to keep record and update all non-dominated solutions. When the entire search finishes, this set consists of all the Pareto-efficient solutions.

When a complete solution is reached, and it is not dominated by the solutions in the non-dominated set, it is added to it. Further, all solutions in this set that are dominated by the new solution are removed, to maintain the non-dominated property.

For efficiency, we implemented the data structure for this non-dominated set as a linked list and hash table hybrid. The hash table allows for quick look-up while the linked list is ordered in increasing makespan values.

Figure 5 illustrates the hybrid data structure. S1, S2, S3, and S4 are schedules in the non-dominated set. The blue and red boxes represent the linked list and hash table portions

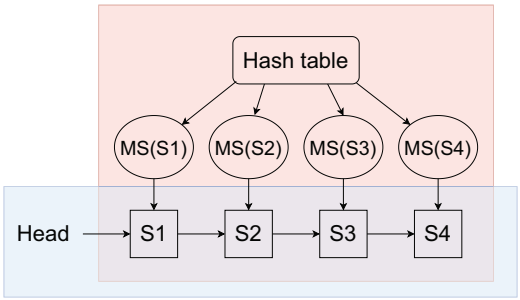


Fig. 5: The hybrid data structure used to store the non-dominating solutions

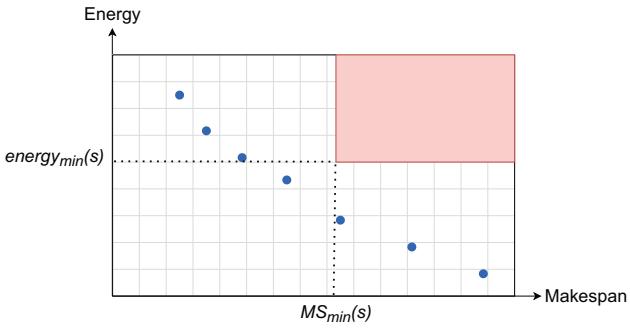


Fig. 6: A visualization of the bounding strategy.

respectively. It is important to note that the hash values directly refer to the linked list nodes, allowing us to transition from searching in the hash table to the linked list seamlessly. This avoids the drawback of longer search times in linked lists (in certain situations).

G. Bounding

Bounding is critical to the performance of the algorithm. This is exacerbated due to the larger search space caused by speed scaling and bi-criteria optimization. As the goal of the research is to find optimum scheduling solutions, bounding should be done with extra care so as to eliminate even the smallest chance of a viable Pareto-efficient solution being disregarded. This constraint prevents the adoption of some of the existing bounding (or pruning) techniques, which leaves us with a simple, but effective strategy.

The novel bounding strategy is implemented within the ordering phase, as there is no information about speed scaling yet available in the allocation phase. As each ordering state is a partial solution, it has a range of possible complete solutions. This means that it has a range of values for possible makespan and energy values.

In order to claim that a certain state (and the sub-tree rooted in it) is not worth exploring, we must make sure that all possible complete solutions associated with the state s are dominated. We assure this by considering the minimum possible makespan ($MS_{min}(s)$) and energy ($E_{min}(s)$) that any complete state based on state s can have. If the solutions in the current non-dominated set dominate $MS_{min}(s)$ and $E_{min}(s)$ it can be discarded. Figure 6 illustrates this effect. The figure shows a non-dominated set as blue dots. The red box represents the range of solutions of a partial state s. If ($MS_{min}(s)$, $E_{min}(s)$) is dominated, all solutions within the red box are dominated. We can observe that ($MS_{min}(s)$, $E_{min}(s)$) is dominated, therefore state s will be discarded and will not be explored further.

$MS_{min}(s)$ is calculated according to (9), where $ordered(s,p)$ and $unordered(s,p)$ refer to the ordered and unordered tasks allocated to processor p in state s. sf_{min} refers to the minimum speed factor in SF.

$$MS_{min}(s) = \max_{p \in P}\{\max_{n_o \in ordered(s,p)} finish_o + \sum_{n_u \in unordered(s,p)} w_u \cdot sf_{min}\} \quad (9)$$

Calculating minimum energy consumption $E_{min}(s)$ of a state s is more complicated. As we have two components of energy, we need to minimize those independently to make sure we are not overestimating energy consumption. The minimum energy of a state is the sum of its minimum dynamic energy $dynamic_{min}(s)$ and minimum static energy $static_{min}(s)$.

Calculating the minimum dynamic energy can be done by making sure all unordered tasks are assumed to be operating at the lowest speed, thus using minimal energy. All ordered tasks are operated at the specified speed. (10) details the calculation of $dynamic_{min}(s)$ of a state s. $all\_ordered(s)$ and $all\_unordered(s)$ refer to the set of ordered and unordered tasks across all processors in state s. sf_{max} refers to the maximum speed factor in SF.

$$dynamic_{min}(s) = \sum_{n_o \in all\_ordered(s)} \frac{w_o}{s_o^2} + \sum_{n_u \in all\_unordered(s)} \frac{w_u}{(sf_{max})^2} \quad (10)$$

In order to obtain the minimum static energy, the minimum makespan should be used (a higher makespan results in higher static energy). Depending on the static energy model, we can choose either all processors or all active processors to calculate static energy. Equation (11) details the minimum static energy calculation. AE, EU, EO, and $none$ refer to their relevant static energy models.

$$static_{min}(s) = MS_{min}(s) \cdot \omega \cdot \left\{ \begin{array}{ll} |P|, & AE \\ |P_{active}|, & EU \\ |P_{active}|, & EO \\ 0, & none \end{array} \right\} \quad (11)$$

Apart from bounding, the original AO state space used multiple pruning and optimization techniques to improve performance. We were able to use most of them within our

research as well, including identical task removal, fixed task order, and graph reversal [15].

V. Evaluation methodology

Our evaluation methodology is similar to that of [10]. The first step is to create a set of task graphs. Three characteristics were used in determining the variety of graphs, which have been shown to be impactful in schedule diversity. The characteristics chosen are graph structure, number of tasks, and computation-to-communication ratio (CCR). CCR is the ratio between the sum of the task weights to the sum of the communication costs. We chose four graph structures (fork-join, random, stencil, series-parallel), two graph sizes (10 and 12 tasks), and three CCR values (0.1, 1.0, 10.0), providing us with 24 unique graphs, summarized in Table I.

TABLE I: Graph characteristics

Graph Structure	No. of Tasks	CCR
Fork-Join	10	0.1
Random	12	1.0
Stencil		10.0
Series-parallel		

The second step is to choose the target system parameters. As introduced in section III, there are three parameters to be considered: static energy model (SEM), speed set, and number of processors. We used four SEMs (none, always everywhere (AE), exclude unused (EU), exclude outside (EO)), three processor counts (2, 4, 6) and two different speeds sets, which are Linear ($\{0.8, 1.6, 2.4, 3.2\}$) and High-speed ($\{0.5, 2.5, 3.4\}$). These parameters choices are summarized in Table II.

This provided us with 24 unique configurations. Each configuration was executed with each graph, resulting in 576 problem instances (24 graphs $\times$ 24 configurations).

TABLE II: Target system parameters

SEM	Speed Set	No. of Processors
None	Linear (0.8, 1.6, 2.4, 3.2)	2
Always Everywhere	High-speed (0.5, 2.5, 3.4)	4
Exclude Unused		6
Exclude Outside		

The solution was implemented in Java (using Java 11) using an existing implementation of the AO state space with state-space search [15]. A large timeout was set for the execution, as the search is computationally expensive. Upon the results of a few test runs, 3 hours and 5 hours were chosen as the timeouts for the 10-task and 12-task problem instances respectively. Each problem instance was run in its own JVM instance, to minimize influence between problem instances. The experiments were run on a server machine with dual AMD EPYC 7702 64-Core Processors with 512 GB of RAM. The static energy power ω was set at a tenth of the power when operating at maximum frequency, i.e. $\omega = \frac{1}{(sf_{min})^3 \cdot 10}$.

The following data was collected for the evaluation of our 576 problem instances: The Pareto-efficient set (at completion or timeout), the total number of solutions (at completion or timeout), execution time, and a timeout indicator.

TABLE III: Number of timeout instances across different graph types

Graph type	10 tasks	12 tasks
Fork-join	27 (37.5%)	72 (100%)
Random	10 (13.9%)	72 (100%)
Series-parallel	6 (8.3%)	29 (40.3%)
Stencil	1 (1.4%)	34 (47.2%)

VI. Results & discussion

Upon obtaining the results from the experiments, we analyzed the data from a multitude of angles. The interesting results and analysis are explained in this section, along with a discussion of possible insights to be gained from them.

We analyzed the distribution of timeouts in problem instances across multiple facets. We found interesting observations when considering graph types. Table III shows the total number of timeout instances across graph types and task counts.

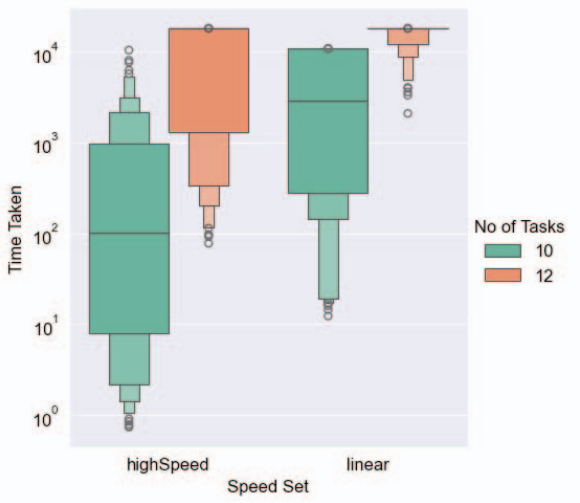
We notice that fork-join has the largest number of timeouts. This can be attributed to the freedom of ordering that fork-join enjoys, leading to a larger number of ordering states to explore. Series-parallel and stencil have minimal timeouts as the task graphs are highly constrained. As predicted in IV-E, ordering states are seen to have contributed heavily to the performance of the solution. We can also notice that a larger number of problem instances have timed out with 12 tasks when compared with 10 tasks. This is in spite of a larger timeout given to 12 task instances (5 hours vs 3 hours). This is testament to the exponentially increasing computational cost of the solution.

We also analyzed the time taken to complete the scheduling algorithm, and the results are shown in Figure 7. It shows boxen plots (a.k.a. letter value plots, a variation of box plots) with time taken on the y-axis and the considered facets on the x-axis. The y-axis uses a log scale for better visibility of diverse values. The plots consider all problem instances, regardless of whether they have timed-out or not. Figure 7a uses all problem instances and splits them across the two speed sets. Figure 7b takes the same problem instances and splits them across the four different graph types. There are separate boxen plots for the different numbers of tasks as well.

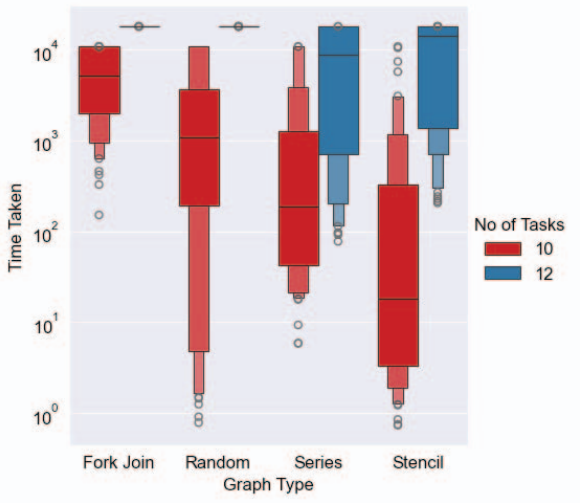
Comparing the speed sets, it is clear that the problem instances using the linear speed set have taken more time than those using the high-speed speed set. This may be attributed to the difference between the number of different speeds in the two sets.Increasing the number of speeds causes an increase in the number of states in the state space.

When comparing the different graph structures, it can be observed that fork-join, random, series-parallel, and stencil have decreasing time taken. This follows the same pattern as the timeout analysis, further reinforcing those insights. The same patterns are also seen when comparing 10 task with 12 task instances.

We also analyzed the total number of Pareto-efficient solutions across different dimensions. We only wanted to



(a) Speedset



(b) Graph type

Fig. 7: Time taken to completion with different facets

consider optimal schedules for this analysis, so all timed-out problem instances were discarded. We noticed an interesting pattern among different static energy models, which is shown in Figure 8. It shows boxen plots of the total number of solutions on the y-axis and the static energy models on the x-axis. The results are also split by the number of tasks.

The number of solutions is an indication of the granularity of the solutions. We can observe that with no static energy, there was a higher number of total solutions when compared to when static energy was present. This indicates the fine-grained solutions in terms of makespan-energy trade-off that can be obtained when static energy is not considered. Task graphs with 12 tasks have a higher number of solutions when compared with 10 tasks. This is a reasonably obvious observation as the number of possible arrangements increases with the number of tasks.

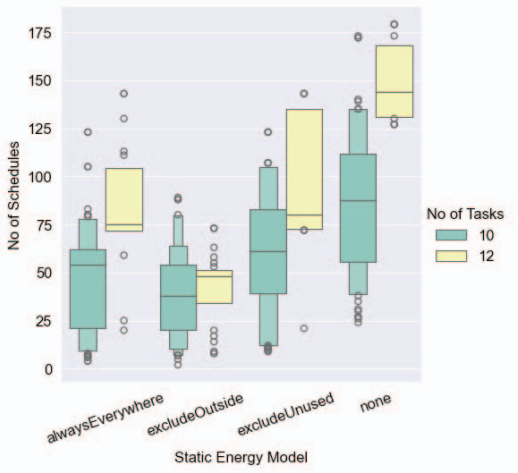


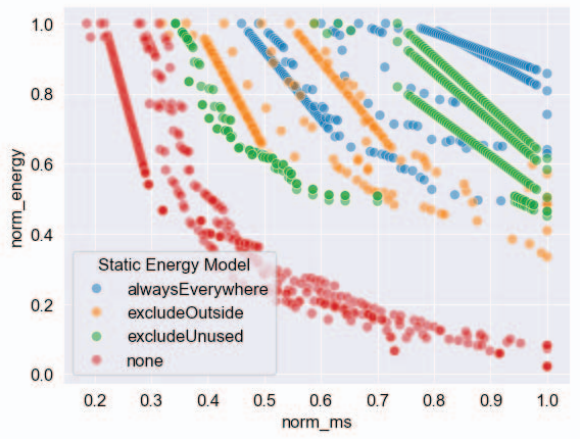
Fig. 8: Number of total solutions with each static energy model

We analyzed the shape and position of Pareto-efficient sets in order to gather insights into how they interact with various problem settings. In order to accomplish this, we grouped all problem instances by task graph type and number of tasks, and then created scatterplots with all the Pareto-efficient sets related to the task graphs. Only non-timeout problem instances were used in this analysis since we are interested in optimal schedules. We analyzed different problem settings for patterns and we noticed that the static energy models have an interesting interaction with Pareto-efficient fronts. Figure 9 are two such scatterplots. Figure 9a is the set of Pareto-efficient solutions for the series-parallel graph with 10 tasks and 1.0 CCR. Figure 9b shows the same visualization for the stencil graph with 12 tasks and 1.0 CCR. These two graphs were chosen due to their low number of timeouts and consistent solution counts. The plots show normalized energy on the y-axis and normalized makespan on the x-axis (normalizing is dividing by the maximum value). The values are normalized to bring all solutions into a single space for better comparisons.

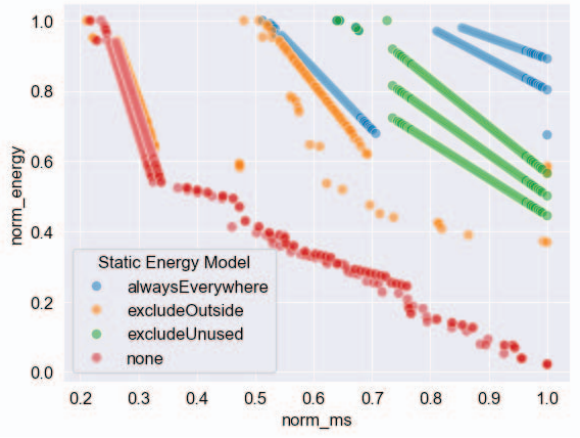
We can notice that with no static energy, the Pareto-efficient front is lower when compared to when static energy is used. This is fairly obvious considering that static energy adds a near-constant energy value to each solution which shifts the whole Pareto-front towards higher makespan and energy values. Among the three other static energy models, ‘always everywhere’ is seen to have to highest Pareto-efficient fronts. This is mainly because it has the highest static energy contribution among all the static energy models.

VII. Conclusion

In this research, we have tried to fill a research gap in a rather unexplored but important problem domain. Energy awareness is a characteristic that is becoming more and more important. With the large amount of processing work happening every day, it is important to consider the cost of it, including the environmental effects. We have created a state space-based solution algorithm for optimal bi-criterion



(a) 10 tasks, series-parallel



(b) 12 tasks, stencil

Fig. 9: How solutions of task graphs vary with static energy model

task scheduling, which has not been done before, as per our knowledge. We have managed to integrate our solution with other state-space search solutions, thus streamlining any future work on bi-criterion state spaces.

We have thoroughly and carefully evaluated our solution across a variety of problem settings, revealing interesting insights. We understood that the structure of the graphs plays a major role in the performance of our solution. Another observation we made is the impact of the static energy model. It is especially impactful within the Pareto-efficient fronts and their granularity.

We understand that our solution algorithm does not scale very well at the moment and is limited to small task graphs. This is a major area for future work. The most important aspect to be improved is the bounding function. With a stronger bound, the number of states that can be pruned can be increased, thus improving performance. Another method of improving performance is to parallelize the algorithm. There are multiple routes that can be taken to parallelize state-space search algorithms, and this can reduce the time taken to complete our solution by a large margin.

References

[1] B. Parhami, *Introduction to parallel processing: algorithms and architectures*. Springer Science & Business Media, 2006.

[2] M. J. Flynn and K. W. Rudd, "Parallel architectures," *ACM computing surveys (CSUR)*, vol. 28, no. 1, pp. 67–70, 1996.

[3] V. Sarkar, *Partitioning and scheduling parallel programs for execution on multiprocessors*. Stanford University, 1987.

[4] M. Drozdowski, *Scheduling for parallel processing*. Springer, 2009, vol. 18.

[5] O. Sinnen, *Task scheduling for parallel systems*. John Wiley & Sons, 2007.

[6] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-based scheduling: applying constraint programming to scheduling problems*. Springer Science & Business Media, 2001, vol. 39.

[7] Y.-K. Kwok and I. Ahmad, "On multiprocessor task scheduling using efficient state space search approaches," *Journal of Parallel and Distributed Computing*, vol. 65, no. 12, pp. 1515–1532, 2005.

[8] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Performance Evaluation*, vol. 67, no. 11, pp. 1155–1171, 2010.

[9] A. S. Pillai, K. Singh, V. Saravanan, A. Anpalagan, I. Woungang, and L. Barolli, "A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems," *Soft Computing*, vol. 22, pp. 3271–3285, 2018.

[10] R. Stewart, A. Raith, and O. Sinnen, "Optimising makespan and energy consumption in task scheduling for parallel systems," *Computers & Operations Research*, vol. 154, p. 106212, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030505482300076X

[11] V. J. Rayward-Smith, "Uet scheduling with unit interprocessor communication delays," *Discrete Applied Mathematics*, vol. 18, no. 1, pp. 55–71, 1987.

[12] U. Hönig and W. Schiffmann, "A comprehensive test bench for the evaluation of scheduling heuristics," in *Proc. 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'04)*. Citeseer, 2004, pp. 437–442.

[13] R. Dietze and G. Rünger, "The search-based scheduling algorithm HP* for parallel tasks on heterogeneous platforms," *Concurrency and Computation: Practice and Experience*, p. e5898, 2020.

[14] T. Davidović and T. Crainic, "Parallel local search to schedule communicating tasks on identical processors," *Parallel Computing*, vol. 48, pp. 1–14, 2015.

[15] M. Orr and O. Sinnen, "Optimal task scheduling benefits from a duplicate-free state-space," *Journal of Parallel and Distributed Computing*, vol. 146, pp. 158–174, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731520303348

[16] J. M. Schutten, "List scheduling revisited," *Operations Research Letters*, vol. 18, no. 4, pp. 167–170, 1996.

[17] T. Davidović, L. Liberti, N. Maculan, and N. Mladenovic, "Towards the optimal solution of the multiprocessor scheduling problem with communication delays," *In Proc. 3rd Multidisciplinary Int. Conf*, 01 2007.

[18] S. Mallach, "Improved mixed-integer programming models for the multiprocessor scheduling problem with communication delays," *J Comb Optim*, vol. 36, p. 871–895, 2018.

[19] S. Roy, R. Devaraj, and A. Sarkar, "Optimal scheduling of PTGs with multiple service levels on heterogeneous distributed systems," in *2019 American Control Conference (ACC)*, 2019, pp. 157–162.

[20] S. Roy, R. Devaraj, A. Sarkar, S. Sinha, and K. Maji, "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)*, 2019, pp. 185–192.

[21] S. Roy, R. Devaraj, A. Sarkar, K. Maji, and S. Sinha, "Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems," *Journal of Systems Architecture*, vol. 105, p. 101706, 2020.

[22] Q. Tang, S. Wu, J. Shi, and J. Wei, "Optimization of duplication-based schedules on network-on-chip based multi-processor system-on-chips," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 826–837, 2017.

[23] Q. Tang, L. Zhu, L. Zhou, J. Xiong, and J. B. Wei, "Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems," *Journal of Parallel and Distributed Computing*, vol. 138, p. 115–127, 2020.
[24] S. Venugopalan and O. Sinnen, "Ilp formulations for optimal task scheduling with communication delays on parallel systems," *Ieee transactions on parallel and distributed systems*, vol. 26, no. 1, pp. 142–151, 2014.
[25] A. Malik, C. Walker, M. O´Sullivan, and O. Sinnen, "Satisfiability Modulo Theory (SMT) formulation for optimal scheduling of task graphs with communication delay," *Journal of Computers and Operations Research*, vol. 89C, pp. 113–126, 2017/2018.
[26] G. Xie, X. Xiao, H. Peng, R. Li, and K. Li, "A survey of low-energy parallel scheduling algorithms," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 1, pp. 27–46, 2021.
[27] K. Huang, X. Jiang, X. Zhang, R. Yan, K. Wang, D. Xiong, and X. Yan, "Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems," *IEEE Access*, vol. 6, pp. 57 614–57 630, 2018.
[28] M. Safari and R. Khorsand, "Energy-aware scheduling algorithm for time-constrained workflow tasks in dvfs-enabled cloud environment," *Simulation Modelling Practice and Theory*, vol. 87, pp. 311–326, 2018.
[29] X. Xiao, G. Xie, C. Xu, C. Fan, R. Li, and K. Li, "Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems," *Journal of computational science*, vol. 26, pp. 344–353, 2018.
[30] P. Eitschberger and J. Keller, "Comparing optimal and heuristic task-graph scheduling on parallel machines with frequency scaling," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 10, 2020.
[31] B. Hu, Z. Cao, and M. Zhou, "Scheduling real-time parallel applications in cloud to minimize energy consumption," *IEEE Transactions on Cloud Computing*, 2019.
[32] K. Huang, X. Jiang, X. Zhang, R. Yan, K. Wang, D. Xiong, and X. Yan, "Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems," *IEEE Access*, vol. 6, pp. 57 614–57 630, 2018.
[33] Y. Qin, G. Zeng, R. Kurachi, Y. Li, Y. Matsubara, and H. Takada, "Energy-efficient intra-task DVFS scheduling using linear programming formulation," *IEEE Access*, vol. 7, pp. 30 536–30 547, 2019.
[34] J. Zhou, J. Sun, P. Cong, Z. Liu, X. Zhou, T. Wei, and S. Hu, "Security-critical energy-aware task scheduling for heterogeneous real-time MPSoCs in IoT," *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp. 745–758, 2019.
[35] I. Ahmad, S. Ranka, and S. Khan, "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, 2008, pp. 1–6.
[36] Y. Lee and A. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374–1381, 2010.
[37] A. Pillai, K. Singh, V. Saravanan, A. Anpalagan, I. Woungang, and L. Barolli, "A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems," *Soft Computing*, vol. 22, no. 10, pp. 3271–3285, 2018.
[38] Z. Zong, A. Manzanares, X. Ruan, and X. Qin, "Ead and pebd: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters," *IEEE Transactions on Computers*, vol. 60, no. 3, pp. 360–374, 2010.
[39] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi, "Adaptive energy minimization of embedded heterogeneous systems using regression-based learning," in *2015 25th international workshop on power and timing modeling, optimization and simulation (PATMOS)*. IEEE, 2015, pp. 103–110.
[40] G. Aupy, A. Benoit, F. Dufossé, and Y. Robert, "Reclaiming the energy of a schedule: models and algorithms," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 11, pp. 1505–1523, 2013.
[41] S. Venugopalan and O. Sinnen, "Memory limited algorithms for optimal task scheduling on parallel systems," *Journal of Parallel and Distributed Computing*, vol. 92, pp. 35–49, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731516000241