Sustainable Computing: Informatics and Systems 1 (2011) 314–328

Contents lists available at SciVerse ScienceDirect

Sustainable Computing: Informatics and Systems

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

Energy efficient task partitioning and real-time scheduling on heterogeneous multiprocessor platforms with QoS requirements

Bader N. Alahmad*, Sathish Gopalakrishnan

University of British Columbia, Vancouver, BC, Canada V6T 1Z4

ARTICLE INFO

Article history:
Received 11 July 2011
Accepted 9 September 2011

Keywords:
Approximation algorithm
Energy
Scheduling
Multiprocessor
Real-time
Quality of service

ABSTRACT

We address the problem of partitioning a set of independent, periodic, real-time tasks over a fixed set of heterogeneous processors while minimizing the energy consumption of the computing platform subject to a guaranteed quality of service requirement. This problem is NP-hard and we present a fully polynomial time approximation scheme for this problem. The main contribution of our work is in tackling the problem in a completely discrete, and possibly arbitrarily structured, setting. In other words, each processor has a discrete set of speed choices. Each task has a computation time that is dependent on the processor that is chosen to execute the task and on the speed at which that processor is operated. Further, the energy consumption of the system is dependent on the decisions regarding task allocation and speed settings.

1. Introduction

We consider a resource allocation problem where we are given a set of heterogeneous processors with discrete speed settings and a set of periodic, independent, real-time tasks. How do we partition the tasks across the available set of processors and choose appropriate speed settings for those processors such that we achieve minimum energy consumption while satisfying some specified quality of service requirement? In this setting, the energy consumed by the complete system depends on the task allocation and on the speeds of the processors. This problem is motivated by two important issues to consider in the design of current and future embedded systems: energy consumption and processor heterogeneity. Furthermore, the emphasis on a system model with completely discrete set of choices is based on architectural considerations as well as empirical evidence that suggests that such a model is indeed the appropriate model to apply.

Energy in embedded systems and especially battery-powered devices is a valuable resource whose expenditure needs to be kept at minimum in order to increase the lifetime of such systems. At the same time, tasks running on those systems should be appropriately serviced according to their computational and timeliness requirements. There are two major contributors to the overall energy consumption in a processor: (i) the *dynamic* power consumption due to switching activities and (ii) the *leakage* power consumption due to leakage current draw in CMOS circuits (Jejurikar et al. [1]) The former depends on the speed at which the processor is operating, while the latter is present whenever the processor is *on*, and is a constant. In addition to the energy consumed by the processor, the total energy consumed by the computing systems depends on the energy consumed by other devices and peripherals in the system (e.g., memory, I/O). This energy consumption is not dependent on the processor speed. This aspect of energy modeling is important because it allows us to capture energy consumption beyond what is specific to the processing units.

Most embedded systems now constitute multiple processors in order to increase the processing throughput. In addition, each processor can be configured to run at a speed (frequency) from a limited set of allowable speeds. Moreover, the processor's operating speed can be varied while the system is running without interrupting task execution. Heterogeneous computing systems consist of multiple processing components having different architectures and computing capabilities, interconnected using different connectivity paradigms. For example, the Nomadik^{TM} platform (Wolf [2]) includes an ARM processor, a video accelerator and an audio accelerator, each of which is itself a heterogeneous multiprocessor. Such systems better meet the demand of applications with diverse requirements, because the computational requirements of a task might differ significantly on different processing elements, and heterogeneous systems allow tasks to be matched to computing elements that better serve their requirements.

* Corresponding author.
E-mail addresses: bader@ece.ubc.ca, baderalahmad84@gmail.com (B.N. Alahmad), sathish@ece.ubc.ca (S. Gopalakrishnan).
URLs: http://blogs.ubc.ca/bader (B.N. Alahmad), http://radical.ece.ubc.ca/sathish (S. Gopalakrishnan).

2210-5379/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.suscom.2011.09.001

In addition to energy, we view the system from a quality-of-service perspective. In general, quality of service is a measure of the maximum error a task might tolerate, or the minimum perceived precision in a dual sense. Quality-of-service is almost always associated with certain cost/precision trade-offs. For example, consider a search engine where the back-end, having received a user query, searches its index database for matching documents, ranks those documents (using some ranking algorithm) and returns the top N documents in ranked order (Baek and Chilimbi [3]). The service provider might consider, for example, executing less cycles running the ranking algorithm for the sake of faster response, in addition to saving energy on the search engine servers. The consequences of such *approximation* of the output of computation become relevant when the QoS metric is clearly defined. We can define the QoS loss metric as the percentage of user requests, compared to the full fledged-service case, that either return the same top N documents but in a different rank order or return different top N documents. Towards the goal of practically enabling such approximations by systematic means, Baek and Chilimbi [3] developed a programming framework called "Green", that allows programmers to approximate loops and expensive functions, and provides statistical QoS guarantees.

In a heterogeneous platform the assignment of tasks to processors may impact the end-user quality of service because certain processor types may be better suited to certain tasks. For example, executing a graphics task on a specialized graphics processor yields better results than performing the same operation on a general-purpose processor. In a system with heterogeneous compute units and multiple speed settings for each processor, a system architect may explore the trade-off between quality of service and energy efficiency. This is the stage of the design process that we target in this article.

To simplify the discussion, we can think of a platform with a fixed number of processors. We need to decide the processor type for each processor from a set of available processor types (permitting heterogeneity in the platform), then selecting a particular speed for a processor and finally assigning tasks to that processor. A task would have a certain worst-case execution time at the selected speed and each instance of a periodic task will consume a certain amount of energy (*active-time energy*) that depends on the type of processor selected and the associated speed of the processor. Additionally when a processor is idle it will consume some energy (*idle-time energy*).

Our energy expenditure model is central to our contribution. We relax many of the impractical assumptions underlying state-of-the-art solutions, including the work of Yang et al. [4], which is the closest to our efforts (see Section 7 for a discussion of related work). In a sense, prior models assume (1) continuity of speed levels on machines, (2) linearity of the worst case execution time (WCET) requirements of tasks with respect to processor speed, (3) constant worst case execution cycles (WCEC) of tasks with respect to speed levels on processors, (4) linear interpolation of the energy expenditure when calculations result in speed levels that are not available on the processor, in case of a discrete processor speed model, and (5) that energy expenditure on a processor depends solely on the total utilization of the processor. These assumptions are agnostic to the fact that different tasks exercise differential, and somewhat arbitrary quality of execution as speed varies on a processor. A consequence of such assumptions is that previous models cannot capture the energy as consumed by devices other than the processor, which is due to nonlinearities of task execution requirements and the overall energy consumption when tasks spend considerable fraction of their time interacting with I/O devices, such as memory and disk (I/O bound tasks). Therefore, those models are very difficult to scale to account for energy expenditure of the compute system as a whole (Section 2.6 provides a more elaborate discussion).

Table 1
Basic math benchmark (single iteration).

Frequency (GHz)	Average power (W)	Execution time (s)
0.8	86.5	32.34
1.6	89.9	31.7
2.1	94	31.61
2.8	100.2	31.75

Table 2
Fast Fourier transform (FFT) benchmark (1000 iterations).

Frequency (GHz)	Average power (W)	Execution time (s)
0.8	84.3	1080.4
1.6	88.96	538.2
2.1	94.78	410.23
2.8	104.11	307.74

To illustrate the abovementioned drawbacks in prior work and to motivate our work with discrete settings, we performed empirical studies using two applications from the MiBench embedded applications benchmark suite by Guthaus et al. [5]: *Basic Math* and *Fast Fourier Transform (FFT)* (with 64 random sinusoids and 65, 536 samples). We executed the former once and the latter 1000 times on an AMD® Phenom$^{\text{TM}}$II X4 925 Quad Core Processor 2.8 GHz, which has discretely variable speeds in the set {800 MHz, 1.6 GHz, 2.1 GHz, 2.8 GHz} per core, and then measured the execution time as well as the energy consumption of each application at each speed step. The Basic Math benchmark includes some I/O operations and we found that speed scaling does not result in corresponding changes in execution time (Table 1). For this benchmark, we also found that energy consumption increases with increases in speed. For the FFT benchmark, which does not include the same amount of I/O as the Basic Math benchmark, execution time decreased as we would expect with an increase in processor speed and it is energy-efficient to run this application at a higher speed (Table 2). Our observations highlight the fact that variations in execution time and energy consumption are different for different applications. These variations are not easily captured by closed-form functions and require discrete modeling.

We present a system and task model (Section 2) that captures system implementations better than earlier models. We focus on determining static speed settings for processors and identifying a mapping between tasks and processors such that the energy expenditure of the system is minimized and the timeliness requirements of tasks respected. In addition, the task allocation scheme should guarantee that the overall QoS satisfies a specified requirement (Section 3).

Due to the practical needs of discrete and arbitrarily structured settings, our solution is combinatorial, and our algorithm employs a blend of matching and enumeration techniques, with dynamic programming being the general algorithmic tool. In summary, our contributions are

1. New model for energy expenditure that alleviates previous impractical assumptions.
2. An FPTAS for the (NP-hard in the ordinary sense) problem of allocating tasks on unrelated parallel machines (heterogeneous platform), where the number of machines is *fixed*, and the goal is to find an assignment of tasks to service classes and simultaneously build a platform from an assortment of given machine types, and then assign tasks (as they are equipped with service classes) to the machines comprising the platform such that the platform expends as minimum energy as possible while the

aggregate quality of service score of the solution assignment is above that of a given quality of service threshold.

3. Provable worst case bounds on the quality of the solution returned by our approximation scheme relative to the optimal solution (produced by the exact, intractable algorithm), expressed in terms of a user-supplied error factor that determines the quality vs. efficiency trade-off.
4. An efficient heuristic (that runs in time quadratic in the number of input tasks) for solving the problem above, equipped with a simulation study to evaluate its efficacy as compared to our FPTAS.

2. System model and notation

2.1. Platform

We have K distinct physical processor types at our disposal, with an unlimited pool of processors of each type. We will use the notation π_k to refer to a physical processor type; therefore, the set $\Pi = \{\pi_1, \ldots, \pi_K\}$ includes all distinct physical processor types, and its elements are represented by integers in, and its elements are represented by integers in $\{1, \ldots, K\}$. Each physical processor type may allow for a discrete number of speed settings. Let $S_k = \{s_1, \ldots, s_{\lambda_k}\}$ be the set of distinct speed levels associated with processor type π_k, where $\lambda_k := |S_k|$. We use $\lambda_{\max}$ to denote the maximum number of speed settings that any physical processor type may permit, i.e., $\lambda_{\max} = \max_{k \in \{1,\ldots,K\}}\{\lambda_k\}$. We introduce the notion of a *logical processor type*, which corresponds to a *unique* physical processor and speed pair; a physical processor running at a certain speed. We do so by creating λ_k logical processor types for physical processor type π_k. Therefore, a logical processor type is defined as $\pi'_k := (\pi_k, s_\ell) \in \{\pi_k\} \times S_k$. There are at most $\lambda = \lambda_{\max} \times K$ logical processor types. The set $\Pi' = \{\pi'_1, \ldots, \pi'_\lambda\}$ includes all logical processor types, and its elements are represented by pairs of integers. Our objective is to construct a platform composed of M heterogeneous physical processors, possibly with identical types, each operating at a certain speed. We refer to *instances* of logical processor types simply as logical processors; accordingly, the platform consists of M logical processors. We denote such a platform composition as a *platform configuration*

$$C = \langle \pi'_1, \ldots, \pi'_M \rangle,$$

where for every $j \in \{1, \ldots, M\}$, π'_j is the logical processor that occupies the j-th slot in C, and is an instance of some logical processor type in Π'[1] (π'_js in C are not necessarily distinct). We can think of a platform as having a fixed number, specifically M, processor "slots", disregarding the order in which processors are arranged. Thus we seek a filling of these slots with "suitable" logical processors. Alternatively, we can represent a platform configuration by the λ-tuple $C = \langle m_1, \ldots, m_\lambda \rangle$, where $m_j \in \{0, \ldots, M\}$ is the number of instances of logical processor type π'_j that C contains, for every $j \in \{1, \ldots \lambda\}$. Using the latter platform configuration encoding, it follows that the number of possible assignments of logical processors to M slots is at most the number of integer solutions to the equation

$$m_1 + m_2 + \cdots + m_\lambda = M.$$

Therefore, we have at most $\Lambda = \binom{\lambda + M - 1}{M}$ possible platform configurations. The alternative platform representation admits a simple combinatorial interpretation of the latter bound: the number of ways M indistinguishable balls can be packed into λ bins.

λ, M and K, and as a consequence Λ, are considered part of the platform model and are treated as constants.

2.2. Task and scheduling model

We consider a task set, Γ, with N periodic, implicit-deadline, real-time tasks such that $\Gamma = \{T_1, \ldots, T_N\}$. This task set needs to be executed on a suitable heterogeneous computing platform. Task T_i recurs every P_i time units and, when executing on logical processor type π'_k has a worst-case execution time of $e_{i,k}$. Since $e_{i,k}$ is specified per logical processor type, it is dependent upon the physical processor type and the frequency at which it is operating. For convenience we will treat P_i to be a positive integer. Correspondingly, the utilization of T_i when assigned to logical processor type π'_k is $u_{i,k} = e_{i,k}/P_i$.

In this discussion, for simplicity, we restrict our attention to the partitioned scheduling of real-time tasks on the heterogeneous multiprocessor platform with earliest deadline first scheduling at each processor. For a certain platform configuration, say C, $z_{i,j}$ is an indicator variable that is 1 if T_i is assigned to π'_j in C and 0 otherwise. All tasks satisfy their timing requirements if, and only if $U_j = \sum_{i=1}^{N} z_{i,j} u_{i,j} \leqslant 1$ for every π'_j in C, $j \in \{1, \ldots, M\}$.

We also note that this work easily applies to the situation when tasks are simply given a fraction of a processor's bandwidth, as may be the case when virtual machines are scheduled in a cloud computing environment.

2.3. Energy model

To account for the energy consumed by the processor and by other associated units (memory, I/O, etc.), our energy model has an active-time energy component and an idle-time energy component. For each physical processor type π_k we have an idle time power consumption, $p_{idle,k}$ (which, of course, all of its λ_k logical processor types inherit) that needs to be accounted for every time unit that a processor is idle. For each logical processor type and task pair, (π'_k, T_i), we denote the energy consumed by executing an instance of T_i on π'_k in a specific time interval as $W_{i,k}$. In its simplest form, $W_{i,k}$ can be defined as $W_{i,k} = u_{i,k} \tau p_{dyn,k}$, where τ is the interval of energy measurement and $p_{dyn,k}$ is the dynamic power consumption of logical processor type π'_k. Given its dependence on the dynamic power consumption of a processor operating at a certain speed, $W_{i,k}$ can be interpreted as a penalty that task T_i incurs as it executes on logical processor type π'_k. We, however, do not restrict $W_{i,k}$ to be of the latter form. Since $W_{i,k}$ can include additional parameters that capture energy consumption at the system level, we treat $W_{i,k}$ as a single quantity.

Depending on the platform configuration, C, and the mapping of tasks to processors, the energy consumed by the system in an interval of length τ, where τ consists of active intervals and idle intervals on the different processors, can be computed as

$$E_C = \sum_{i=1}^{N} \sum_{j=1}^{M} \lfloor \frac{\tau}{P_i} \rfloor z_{i,j} W_{i,j} + \tau \sum_{j=1}^{M} (1 - U_j) p_{idle,j}, \qquad (1)$$

where $\lfloor \tau/P_i \rfloor$ is the number of instances of task T_i that arrive during time interval τ. The first term represents the active-time energy consumption and the second term represents the idle-time energy consumption.

The basic task set consists of periodic tasks and therefore it is sufficient to study and minimize the energy consumed in an interval of length $L = \mathrm{LCM}(P_1, \ldots, P_N)$ because all activity repeats in an

[1] Note that the index j in π'_j as it appears in C refers to the order in C; index j might map to a different logical processor type index in Π'.

identical manner every L time units. Since $P_i \mid L$ for every $i \in \{1, \ldots, N\}$, Eq. (1) becomes

$$E_C = \sum_{i=1}^{N}\sum_{j=1}^{M}\frac{L}{P_i} z_{i,j} W_{i,j} + L\sum_{j=1}^{M}(1-U_j)p_{idle,j}. \tag{2}$$

2.4. Quality-of-service model

We denote the quality of service derived (or delivered) by a task T_i when it is assigned to logical processor type π'_k using a real number $r_{i,k}$. We can also interpret $r_{i,k}$ as a reward obtained by the particular task-to-processor mapping. Our model allows for further generality by allowing the reward to depend on the logical processor type. The aggregate quality of service delivered by the system, or equivalently the reward obtained by the system, is a function of the different $r_{i,k}$ values. *We use a simple sum to describe the aggregate reward although other functions can be accommodated by our framework.*

2.5. Task set encoding

According to the system model above, we encode the requirements of each task T_i in a set of triples (options) O_i, where vectors in O_i are defined as $O_{i,k} = \langle \pi'_k, u_{i,k}, r_{i,k}, W_{i,k}\rangle$. Task T_i is said to be *defined on logical processor type* π'_k if there exists an integer k such that $\pi'_k \in O_{i,k}$. In other words, a task might not be allowed to execute on some logical processor types, for which its execution requirements will not be given. Accordingly, an instance I of our problem is represented as $I = \left(\bigcup_{i=1}^{N}\{O_{i,k}\}_{k=1}^{\lambda}, P_i, Q\right)$. If we assume that rational numbers of the form $x = p/q$, $p, q \in \mathbb{Z}$, $q \neq 0$ are represented as a pair (p, q), then we will need $\mathcal{O}(\log p + \log q)$ space to encode x in binary. Denote as $\texttt{SIZE}(I)$ the number of bits required to represent an instance I in binary. Then

$$SIZE(I) \leqslant N + \sum_{i=1}^{N}\log_2 P_i + \log_2 Q + \sum_{i=1}^{N}\sum_{k=1}^{\lambda}(\log_2 \pi'_k + SIZE(u_{i,k}) + SIZE(r_{i,k}) + SIZE(W_{i,k})),$$

assuming $P_1, \ldots, P_N$ and Q are integers.

2.6. Notes regarding assumptions

Assumption 1. Power ratings include leakage (static) power consumption.

We assume that the leakage power consumption p_{leak} is implicit in both $W_{i,k}$ and p_{idle}. For instance, when *a* processor is idle, its power consumption can be modeled in its simplest form as $p_{idle} = \min_k\{p_{dyn,k}\} + p_{\text{leak}}$, where $p_{dyn,k}$ is the dynamic power rating at some speed, say s_k. Nevertheless, more sophisticated idle power-saving models are employed in current technologies (Intel® Core™ i7 Series incorporates advanced idle modes [6]). This assumption, however, makes the model clearer and simplifies the analysis, without degrading the accuracy of the model.

This entails that no assumptions exist on how task utilizations scale with different speeds. Consider the relationship between the speed of the processor and the execution requirements of tasks. Suppose that some task is known to require a worst case utilization of u_k when running on a processor operating at speed s_k. If the task is to run on a different speed s_ℓ, where $s_k \neq s_\ell$, on the same processor, then a seemingly intuitive way to adjust the execution requirement of this task is to scale u_k linearly as $\hat{u}_k = u_k s_k / s_\ell$. The latter scaling is naïve for many reasons. First, assuming that the WCEC of a task is constant and does not change with processor speed, executing a task at a higher speed will shorten its utilization. It is evident from Eq. (2) that decreasing the utilization increases both the idle and the leakage power consumption. Second, the naïve approach assumes that the WCEC is constant with respect to speed. Indeed, this is inaccurate because a task might access other devices throughout the course of its execution, such as memory and disk. Such devices typically run at bus frequencies that might be divergent from the processor's frequency (usually constant or have limited configurable speeds compared to those of the processor). Therefore, if the device bus has constant latency, then increasing the processor speed will increase the discrepancy between the speed of the processor and that of the device bus. Accordingly, the task will spend more cycles waiting for the device (doing no useful work). Therefore, increasing the processor speed might in fact increase the WCEC required by a task and therefore increase the WCET overall (which is not what one would expect from increasing the speed of the processor). The latter situation becomes more noticeable if the task is I/O-bound. Since energy depends on the WCET, the energy expenditure might be significantly larger than that captured by the naïve method.

Finally, suppose that devices can operate in three states, namely the *active*, *standby* and the *off* state. Further, assume that all devices that a task requires are turned on in the standby state at the instance the task starts execution. Slowing down the processor will cause devices to wait longer in the standby state to be accessed by tasks requiring them, thus increasing the device standby energy. If the task is I/O-bound, then the device idle energy might dominate the energy consumption of the task and the overall energy expenditure might increase.

Assumption 2. Each processor adopts a uniprocessor scheduling policy that is capable of utilizing the processor up to 100%.

An example of such a policy is *Earliest Deadline First* (EDF) (Liu and Layland [7]). This assumption, alongside the previous assumption, guarantee that each processor is a bin of unit capacity. To see this, let D_i, $D_i \leqslant P_i$, denote the *relative deadline* of task T_i. Let $\texttt{demand}(t) > 0$ be the worst case execution requirements of a set of N periodic tasks that arrive and must complete within a contiguous interval of length t. Baruah et al. [8] derived the following expression for the execution-time demand of the task set on one processor

$$demand(t) = \sum_{i=1}^{N}\left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor e_i. \tag{3}$$

Further, they derived sufficient and necessary conditions for the schedulability of N periodic tasks on one processor that employs EDF. Specifically, the task set is schedulable iff $\texttt{demand}(t) \leqslant t$. In our work we assume that tasks have *implicit deadlines*, i.e., $D_i := P_i$, and since our energy measurement interval is L, the schedulability condition on one processor becomes

$$demand(L) = \sum_{i=1}^{N}\left\lfloor \frac{L}{P_i} \right\rfloor e_i \leqslant L. \tag{4}$$

Moreover we have

$$\sum_{i=1}^{N}\left\lfloor \frac{L}{P_i} \right\rfloor e_i \leqslant \sum_{i=1}^{N}\frac{L}{P_i} e_i = L\sum_{i=1}^{N} u_i \leqslant L,$$

from which we get that $\sum_{i=1}^{N} u_i \leqslant 1$, which is our unit-capacity bin condition. Further, we assume that the uniprocessor scheduling policy is *work-conserving*, that is, the scheduler always dispatches a task that is ready for execution as soon as the processor becomes idle. The extension to other similar scheduling policies that provide utilization bounds is straightforward.

Assumption 3. Problem settings are discrete.

The set of processor speeds is arbitrarily structured, as well as the set of penalties incurred by different processor speeds. Further, we make no assumptions regarding the structure of the set of rewards obtained by tasks as they execute on different processors. More specifically, we neither make assumptions about the nature of the reward functions – whether they are uniform, concave, etc. – nor about the relationship of rewards to processors and speeds. In other words, tasks are heterogeneous with respect to their power characteristics; different tasks may be associated with different power functions.

3. Problem formulation

For the system model above, we wish to establish appropriate pairings between (i) processors and speeds and (ii) tasks and processors, so as to satisfy the following requirements:

1. Each processor is assigned exactly one power rating (speed) for the whole duration of its operation.
2. The total energy expenditure of all processors is at a minimum.
3. A processor should never exceed its capacity.
4. Once assigned to a processor, a task should execute on that processor in its entirety.
5. For a certain aggregate quality of service score Q, a minimum guarantee at the system level should be achieved.

For a certain platform configuration $C = \langle \pi'_1, \ldots, \pi'_M \rangle$, we formulate the task assignment problem as a mathematical program as follows: Given the task set encoding $O = \{O_i\}_{i=1}^{N}$ of an instance I, we represent task T_i's settings per C as follows:

Let $z_{i,j}$ be the following indicator variable:

$$Z_C = \begin{pmatrix} Z_{1,1} & \cdots & Z_{1,M} \\ \vdots & \ddots & \vdots \\ Z_{N,1} & \cdots & Z_{N,M} \end{pmatrix}$$

Then its associated binary $N \times M$ *assignment* matrix, which is to be determined, is:

$$Z_C = \begin{pmatrix} z_{1,1} & \cdots & z_{1,M} \\ \vdots & \ddots & \vdots \\ z_{N,1} & \cdots & z_{N,M} \end{pmatrix}$$

The utilization matrix is given by:

$$U_C = \begin{pmatrix} u_{1,1} & \cdots & u_{1,M} \\ \vdots & \ddots & \vdots \\ u_{N,1} & \cdots & u_{N,M} \end{pmatrix}$$

where

$$u_{i,j} = \begin{cases} u_{i,k} & \text{if } \exists k \text{ s.t. } \pi'_j = \pi'_k \text{ in some option } O_{i,k}, \\ 0 & \text{otherwise.} \end{cases}$$

The reward matrix is:

$$R_C = \begin{pmatrix} r_{1,1} & \cdots & r_{1,M} \\ \vdots & \ddots & \vdots \\ r_{N,1} & \cdots & r_{N,M} \end{pmatrix}$$

where

$$r_{i,j} = \begin{cases} r_{i,k} & \text{if } \exists k \text{ s.t. } \pi'_j = \pi'_k \text{ in some option } O_{i,k}, \\ 0 & \text{otherwise.} \end{cases}$$

The energy matrix is given by:

$$\mathcal{W}_C = \begin{pmatrix} W_{1,1} & \cdots & W_{1,M} \\ \vdots & \ddots & \vdots \\ W_{N,1} & \cdots & W_{N,M} \end{pmatrix}$$

where

$$W_{i,j} = \begin{cases} W_{i,k} & \text{if } \exists k \text{ s.t. } \pi'_j = \pi'_k \text{ in some option } O_{i,k}, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, define the idle power vector as:

$$P_{\text{idle},C} = \left[p_{\text{idle},1} \quad \cdots \quad p_{\text{idle},M} \right]^T$$

where $p_{\text{idle},j} = p_{\text{idle},k}$ if $\pi'_j = \pi'_k$ for some k. The mathematical program can be written as:

$$\text{minimize} \quad E = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{L}{P_i} z_{i,j} W_{i,j} + L \sum_{j=1}^{M} (1 - U_j) p_{\text{idle},j} \tag{5}$$

$$\text{subject to} \quad \sum_{j=1}^{M} z_{i,j} = 1 \qquad \forall i \in \{1, \ldots, N\} \tag{6}$$

$$\sum_{i=1}^{N} u_{i,j} z_{i,j} \leqslant 1 \qquad \forall j \in \{1, \ldots, M\} \tag{7}$$

$$\sum_{j=1}^{M} \sum_{i=1}^{N} r_{i,j} z_{i,j} \geqslant Q$$

$$z_{i,j} \in \{0, 1\} \qquad \forall i \in \{1, \ldots, N\}, \forall j \in \{1, \ldots, M\}$$

where Q is the minimum aggregate QoS score that the system should achieve. Constraint (6) says that a task should be assigned to a single processor, and constraint (7) guarantees that a processor's capacity is not exceeded. Therefore, the goal is to find the matrix Z that represents an assignment of tasks to processors that minimizes the energy expenditure of configuration C.

Intractability: The solution points represented by matrix Z_C are required to be binary, for we do not split a task across processors, so fractional assignments are not of our interest. This renders our mathematical program computationally intractable. In fact, it is not difficult to show that this problem is NP-hard by reduction from the PARTITION problem (restrict the platform to have two processors), which was shown by Garey and Johnson [9] to be NP-Complete. Therefore a polynomial-time approximation algorithm for solving this problem within a factor of the optimal solution might be our only resort unless P = NP.

4. Assignment of tasks to processors

We resort to a dynamic programming (DP) approach to solve this problem. We show that an exact DP formulation has complexity that is exponential in time and space requirements. We then use the exact DP to obtain a modified DP that is computationally tractable and is a fully polynomial-time approximation scheme (FPTAS).

4.1. Exact and optimal dynamic program formulation

For a certain platform configuration of the form $C = \langle \pi'_1, \ldots, \pi'_M \rangle$, we equip each task T_i with a set of states $\mathcal{S}_i = \{\varphi_1, \ldots, \varphi_{|\mathcal{S}_i|}\}$. Each state $\varphi \in \mathcal{S}_i$ is an M-dimensional vector that encodes a partial feasible schedule from T_1 up to T_i on every π'_j in C. More precisely, a state has the form

$$\varphi \doteq \langle (Y^{\varphi}_{i,1}, E^{\varphi}_{i,1}), \ldots, (Y^{\varphi}_{i,M}, E^{\varphi}_{i,M}) \rangle,$$

where, for every pair $(Y^{\varphi}_{i,j}, E^{\varphi}_{i,j})$, $j \in \{1, \ldots, M\}$, $Y^{\varphi}_{i,j}$ is a set that contains information about tasks assigned to processor π'_j, i.e., $Y^{\varphi}_{i,j} \doteq \{\langle T_\ell, u_{\ell,j}, r_{\ell,j}\rangle\}_\ell$ for every task assigned to π'_j, where $1 \leqslant \ell \leqslant i$. $E^{\varphi}_{i,j}$ is the energy expenditure of π'_j with respect to the schedule $Y^{\varphi}_{i,j}$. Let $U^{\varphi}_{i,j}$ be the total utilization (workload) on π'_j

$$U^{\varphi}_{i,j} = \sum_{\langle T_\ell, u_{\ell,j}, r_{\ell,j}\rangle \in Y^{\varphi}_{i,j}} u_{\ell,j},$$

then $E^{\varphi}_{i,j}$ is computed as

$$E^{\varphi}_{i,j} = \sum_{\langle T_\ell, u_{\ell,j}, r_{\ell,j}\rangle \in Y^{\varphi}_{i,j}} \frac{L}{P_\ell} W_{\ell,j} + L(1 - U^{\varphi}_{i,j}) p_{\mathrm{idle},j},$$

in accordance with Eq. (5).

The input is a set of task option encodings O_i and the desired quality score Q. In order to reflect T_i's parameters with respect to the configuration in consideration C, each O_i is processed in a way similar to the definition of U_C and R_C matrices in the mathematical program above to produce X_i. It is easy to see that this input processing is polynomial in the input size.

We utilize the framework developed by Woeginger [10] for formulating dynamic programs as follows. Let the initial state be $\mathbf{0} = \langle (\emptyset, 0), \ldots, (\emptyset, 0)\rangle$, where $|\mathbf{0}| = M$. We start with the initial state space $\mathcal{S}_0 = \{\mathbf{0}\}$, and generate new states in the state space $\mathcal{S}_i$ from $\mathcal{S}_{i-1}$ by means of a finite set of mapping functions $\mathcal{F}_C = \{f_1, \ldots, f_M\}$. Each $f_j \in \mathcal{F}_C$ specifies how a task is packed into the j-th processor π'_j in configuration C and generates a new state from it. For a certain $\varphi \in \mathcal{S}_{i-1}$ and task T_i, $\varphi' = f_j(C, X_i, \varphi)$ is the state that results from φ by updating the j-th pair $(Y^{\varphi}_{i-1,j}, E^{\varphi}_{i-1,j})$ in φ as follows: the triplet $\langle T_i, u_{i,j}, r_{i,j}\rangle$ is added to the schedule $Y^{\varphi}_{i-1,j}$ to produce $Y^{\varphi'}_{i,j}$ and the value of the new energy expenditure of the updated schedule is computed according to $u_{i,j}$ as follows

$$\begin{aligned} E^{\varphi'}_{i,j} &= \sum_{\langle T_\ell, u_{\ell,j}, r_{\ell,j}\rangle \in Y^{\varphi}_{i-1,j}} \frac{L}{P_\ell} W_{\ell,j} + \frac{L}{P_i} W_{i,j} + L(1 - (U^{\varphi}_{i-1,j} + u_{i,j}) p_{\mathrm{idle},j} \\ &= E^{\varphi}_{i-1,j} + \frac{L}{P_i} W_{i,j} - L u_{i,j} p_{\mathrm{idle},j}, \end{aligned} \tag{8}$$

where $E^{\mathbf{0}}_{0,j} = p_{\mathrm{idle},j}$ is the initial energy that processor π'_j in configuration C consumes when it is not assigned any tasks.

For each f_j there exists a mapping $h_{f_j} \in \mathcal{H}_C$ that serves as a tool to rule out infeasible assignments. We define h_{f_j} as

$$h_{f_j}(C, X_i, \varphi) = \begin{cases} U^{\varphi}_{i-1,j} + u_{i,j} - 1 & \text{if } u_{i,j} > 0, \\ \infty & \text{otherwise.} \end{cases} \tag{9}$$

A new state $\varphi' \in \mathcal{S}_i$ can be produced from $\varphi \in \mathcal{S}_{i-1}$ if, and only if $h_{f_j}(C, X_i, \varphi) \leqslant 0$. This definition of h_{f_j} captures both the requirements that a processor's capacity should never be exceeded as stated in inequalities (7), and that a task must be assigned to at least one processor over which it is defined for a schedule to be feasible. A platform configuration will therefore be invalid if there is at least one task that cannot be assigned to any processor in this configuration, over all states in the state space of the preceding task. This can occur for a task T_i and configuration C either because T_i is not defined on any π'_j in C, i.e., $u_{i,j} = 0$, and for which $h_{f_j}(C, X_i, \varphi) = \infty$ for all $\varphi \in \mathcal{S}_{i-1}$, or because assigning T_i to π'_j will violate its capacity constraint, i.e., $U^{\varphi}_{i-1,j} + u_{i,j} > 1$, and for which $h_{f_j}(C, X_i, \varphi) = U^{\varphi}_{i-1,j} + u_{i,j} - 1 > 0$, for all $\varphi \in \mathcal{S}_{i-1}$.

Since $\mathcal{S}_N$ contains all feasible schedules of all N tasks across the processors in a platform configuration, it follows that there should exist an optimal schedule $\varphi^* \in \mathcal{S}_N$ for which $E(\varphi^*)$ is minimum, i.e., $\textsc{Opt} = \mathrm{E}(\varphi^*) = \textsc{min}\,\{\mathrm{E}(\varphi) : \varphi \in \mathcal{S}_N\}$, where $E(\varphi) = \sum_{j=1}^{M} E^{\varphi}_{N,j}$.

Algorithm ScheduleExact considers all configurations and, for each task T_i, generates all distributions of utilizations to processors φ from those of T_{i-1} and eliminates infeasible ones according to h_{f_j}. For each configuration, it finds the state in the state space $\mathcal{S}_N$ of the N-th task with the minimum energy expenditure and adds both its energy and reward to the solution space. Finally, the solution space is ordered by non-decreasing energy and the first solution point to achieve the desired QoS score is chosen.

It should be noted that the solution points in the resulting solution space $\mathcal{A}$ form a partially ordered set, where some elements are incomparable. A point (E_q, R_q) is *energy inefficient with respect to reward* if there exists at least one point (E_ℓ, R_ℓ) such that $E_q \geqslant E_\ell$ and $R_q \leqslant R_\ell$. This means that the first solution consumes more energy than the second while attaining less reward. Those points are incomparable, so prior to ordering $\mathcal{A}$, the algorithm eliminates solution points that are energy inefficient with respect to reward to produce a total ordering on the solution space $\mathcal{A}$.

Algorithm 1. ScheduleExact($\langle O_1, \ldots, O_N\rangle$, Q)

```
1 $\mathcal{A} \leftarrow \emptyset$
2 $\mathcal{S}_0 \leftarrow \{\langle(\emptyset, 0), \ldots, (\emptyset, 0)\rangle\}$
3 foreach $C$ of the $\binom{\lambda + M - 1}{M}$ configurations do
4   invalidConfiguration ← true
5   For $i \leftarrow 1$ to $N$ do
6     $\mathcal{S}_i \leftarrow \emptyset$
7     foreach $\varphi \in \mathcal{S}_{i-1}$ do
8       Process $O_i$ according to $C$ to generate $X_i$
9       for $j \leftarrow 1$ to $M$
10        if $h_{f_j}(C, X_i, \varphi) \leqslant 0$ then
11          $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{f_j(C, X_i, \varphi)\}$
12          invalidConfiguration ← false
13        end if
14      end for
15    end foreach
16    if invalidConfiguration is true then
17      Move to the next configuration
18    end if
19  end for
20  Examine every pair of states $\varphi$ and $\varphi' \in \mathcal{S}_N$. If $E(\varphi) \leqslant E(\varphi')$
    and $R(\varphi) \geqslant R(\varphi')$ then remove $\varphi'$ from $\mathcal{S}_N$
21  Sort $\mathcal{S}_N$ by non-decreasing $E(\varphi)$; sort entries with equal
    energy value by non-increasing reward $R(\varphi)$
22  $\hat{\varphi} \leftarrow$ the first $\varphi$ in the sorted $\mathcal{S}_N$ such that $R(\varphi) \geqslant Q$
23  if no $\varphi$ in $\mathcal{S}_N$ achieves $R(\varphi) \geqslant Q$ then
24    Move to the next configuration
25  end if
26  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(C, \hat{\varphi})\}$
27 end foreach
28 if all platform configurations are invalid then
29   Report "No feasible schedule exists" and terminate
30 end if
31 Examine every pair $(C, \varphi)$ and $(C', \varphi') \in \mathcal{A}$. If $E(\varphi) \leqslant E(\varphi')$ and
   $R(\varphi) \geqslant R(\varphi')$ then remove $(C', \varphi')$ from $\mathcal{A}$
32 Sort $\mathcal{A}$ by non-decreasing $E(\varphi)$; sort entries with equal energy
   value by non-increasing reward $R(\varphi)$
33 return the first $(C, \varphi)$ in the sorted $\mathcal{A}$
```

Proposition 1. ScheduleExact *is exponential in both time and space.*

Proof. The algorithm starts with the initial state $\mathcal{S}_0 = \{\langle(\emptyset, 0), \ldots, (\emptyset, 0)\rangle\}$, so $|\mathcal{S}_0| = 1$. Consider some task assignment to a particular configuration C. At each iteration i of the algorithm,

$1 \leqslant i \leqslant N$, each option $x_i \in X_i$ can generate one or more states from each $\varphi_{i-1} \in \mathcal{S}_{i-1}$, because C might aggregate multiple identical processors that match x_i. Whenever T_i is assigned to a processor, that processor is 'stripped out' because T_i cannot be assigned to the same processor more than once. Therefore the number of possible assignments of a task to a configuration is at most M. This means that the state space expands by a factor of at most M for each task. Thus the total number of states in the i-th iteration is bounded from above by

$$|\mathcal{S}_i| \leqslant M|\mathcal{S}_{i-1}|.$$

The total number of states is

$$\prod_{i=1}^{N} \frac{|\mathcal{S}_i|}{|\mathcal{S}_{i-1}|} = \frac{|\mathcal{S}_N|}{|\mathcal{S}_0|} \leqslant M^N,$$

so

$$|\mathcal{S}_N| \leqslant M^N.$$

Considering all configurations, ScheduleExact will require at least $\mathcal{O}\left(\binom{\lambda + M - 1}{M} M^N\right)$ time and space, which proves the claim. □

4.2. The fully polynomial-time approximation scheme

The exponential state space generated above can be significantly reduced by observing that some states are 'close' to each other, and close states can be pruned without severely sacrificing optimality.

Define the error factor ϵ to be an input that specifies the desired trade off between the accuracy of the solution and the running time of the algorithm. It can be in the range

$$0 < \epsilon \leqslant 1. \tag{10}$$

We use the technique of Trimming-The-State-Space, introduced by Ibarra and Kim [11] and treated rigorously by Woeginger [10], in order to thin out the state space by merging 'close' states and bring down the size of the state space to a polynomial in N and $1/\epsilon$. Doing so allows us to derive an FPTAS that produces a solution having a value $\widetilde{E}$ whose relative error with respect to the value of the optimal solution E^* is bounded above by the error factor ϵ, i.e., $(\widetilde{E} - E^*)/E^* \leqslant \epsilon$, and therefore the approximate solution is at most $(1+\epsilon)$ far from the optimal solution: $\widetilde{E} \leqslant (1+\epsilon)E^*$.

In order to do so, we need a measure of closeness between states to quantitatively decide on how to prune states so that the error resulting from such pruning is bounded and controlled, which we state in the following definitions.

Definition 1. Let x, y be any real numbers. We say that x and y are Δ-close, where $\Delta \geqslant 1$ is the trimming factor, if $(1/\Delta)x \leqslant y \leqslant \Delta x$.

The following proposition lists some useful and easily provable properties of Δ-closeness.

Proposition 2. *Properties of Δ-closeness*

1. *Δ-closeness is both reflexive and symmetric.*
2. *If x is Δ_1-close to y and y is Δ_2-close to z, then x is $\Delta_1\Delta_2$-close to z.*

We extend the definition above naturally to M-dimensional vectors over $\mathbb{R}^M$.

Definition 2. Let $\mathbf{x}, \mathbf{y}$ be any vectors in $\mathbb{R}^M$. We say that $\mathbf{x}$ and $\mathbf{y}$ are Δ-close, where $\Delta \geqslant 1$ is the trimming factor, if $(1/\Delta)x_j \leqslant y_j \leqslant \Delta x_j$ for every $j \in \{1, \ldots, M\}$.

Now we make Definition 2 specific to the vectors in our application.

Definition 3. Let $\mathcal{S}_i$ be the state space of task T_i. Two states $\varphi, \varphi' \in \mathcal{S}_i$ are Δ-close with respect to energy expenditure, where $\Delta \geqslant 1$ is the trimming factor, if $(1/\Delta)E_{i,j}^{\varphi} \leqslant E_{i,j}^{\varphi'} \leqslant \Delta E_{i,j}^{\varphi}$ for all $j \in \{1, \ldots, M\}$.

We denote two states that are Δ-close with respect to energy as $[E, \Delta]$-close. This means that $E_{i,j}^{\varphi}$ and $E_{i,j}^{\varphi'}$ are 'good' representatives of each other, for all of the M processors in the platform, and either φ or φ' can be kept and the other discarded; the decision of which to keep depends on the application [1]. By pruning states according to the measure above, we can obtain a solution that is 'close-enough' to the optimal while being able to control the error propagation resulting from such pruning.

State space pruning: From now on, we shall denote the unpruned state space as $\mathcal{U}$ and the pruned state space (resulting from refining $\mathcal{U}$) as $\mathcal{P}$. The idea of state space pruning is to partition the state space into groups containing $[E, \Delta]$-close states, and then selecting one state from each non-empty group to enter the state space. We denote such groups as Δ-boxes $\mathcal{B}_k$, where $\mathcal{B}_k \subseteq \mathcal{U}$ for all k (Woeginger [10]).

Consider the state space $\mathcal{U}_i$ generated by T_i. Let $\mathcal{E}_{i,j}$ be a multiset containing the energy values of the j-th coordinate of every $\varphi \in \mathcal{U}_i$

$$\mathcal{E}_{i,j} = \{E_{i,j}^{\varphi} : \varphi \in \mathcal{U}_i\}.$$

In other words, $\mathcal{E}_{i,j}$ is the set of energy values associated with schedules on processor π'_j among all partial feasible schedules up to the i-th task T_i in some platform configuration. Let $E_{i,\max} = \max_j \max\{E_{i,j}^{\varphi} : \varphi \in \mathcal{U}_i\}$. Then $E_{i,\max}$ is the maximum energy value across all processors and states in the state space $\mathcal{U}_i$. Now let $L_i = \lceil \log_\Delta \max(1, E_{i,\max}) \rceil$. Every vector in $\mathcal{U}_i$ is in $[0, \Delta^{L_i}]^M$ and thus $\mathcal{U}_i \subseteq [0, \Delta^{L_i}]^M$. We create $L_i + 1$ intervals, where $\mathcal{I}_0 = [0, 1)$, $\mathcal{I}_k = [\Delta^{k-1}, \Delta^k)$ for all $k \in \{1, \ldots, L_i - 1\}$, and $\mathcal{I}_{L_i} = [\Delta^{L_i - 1}, \Delta^{L_i}]$. Each $E_{i,j}^{\varphi} \in \mathcal{E}_{i,j}$ resides in exactly one interval, and cannot exceed Δ^{L_i}. We partition the energy values in each $\mathcal{E}_{i,j}$ to intervals as defined above, mapping each $E_{i,j}^{\varphi} \in \mathcal{E}_{i,j}$ to the interval to which it belongs. Assuming that $\texttt{SIZE}(E_{i,\max})$ is polynomial in the size of the instance $\texttt{SIZE}(I, \epsilon)$, the number of the thereby constructed intervals will be polynomially bounded in $\texttt{SIZE}(I, \epsilon)$. Further, we claim that for any two states, if for every coordinate j the energy values of both states are in the same interval, the those states are $[E, \Delta]$-close.

Claim 1. *Let $\mathcal{U}_i$ be the state space of task T_i. Let φ, φ' be two states in $\mathcal{U}_i$. If $E_{i,j}^{\varphi}$ and $E_{i,j}^{\varphi'}$ are in the same interval for every $j \in \{1, \ldots, M\}$, then φ and φ' are $[E, \Delta]$-close.*

Proof. Let $E_{i,j}^{\varphi}, E_{i,j}^{\varphi'} \in \mathcal{I}_k = [\Delta^{k-1}, \Delta^k]$ for some $k \in \{1, \ldots, L_i\}$, for all coordinates $j, j \in \{1, \ldots, M\}$. Then

$$\Delta^{k-1} \leqslant E_{i,j}^{\varphi'} \leqslant \Delta^k \quad \forall j \in \{1, \ldots, M\} \tag{11}$$

and

$$\Delta^{k-1} \leqslant E_{i,j}^{\varphi} \leqslant \Delta^k \quad \forall j \in \{1, \ldots, M\}. \tag{12}$$

Write the RHS of (12) as $(1/\Delta)E_{i,j}^{\varphi} \leqslant \Delta^{k-1}$ and the LHS of (12) as $\Delta^k \leqslant \Delta E_{i,j}^{\varphi}$. Substituting the rewritten inequalities into (11) yields

$$\frac{1}{\Delta} E_{i,j}^{\varphi} \leqslant \Delta^{k-1} \leqslant E_{i,j}^{\varphi'} \leqslant \Delta^k \leqslant \Delta E_{i,j}^{\varphi}$$

for every $j \in \{1, \ldots, M\}$. Thus φ and φ' are $[E, \Delta]$-close by our definition of $[E, \Delta]$-closeness (Definition 1). □

[1] The reader may refer to Cormen et al. [12] chapter 35 for an example of pruning as applied to the $\texttt{SUBSET\_SUM}$ problem.

Algorithm SCHEDULEPRUNED is a realization of the idea above. Procedure PRUNE performs state pruning. For every coordinate of every state φ, in addition to the partial schedule $Y^{\varphi}_{i,j}$ and the energy value $E^{\varphi}_{i,j}$ of this schedule, the algorithm maintains the number of the interval to which $E^{\varphi}_{i,j}$ belongs, which we will denote as $\ell^{\varphi}_{i,j}$, where $\ell^{\varphi}_{i,j} \in \{0, \ldots, L_i\}$.

The pruning procedure will admit a single state from every $\mathcal{B}_k$ where $\mathcal{B}_k \cap \mathcal{U} \neq \emptyset$ into the pruned state space $\mathcal{P}$. Specifically, we choose the state with maximum reward over all states in the Δ-box in consideration to enter the pruned state space. Therefore, the number of states in the pruned state space is exactly the number of non-empty Δ-boxes. As a matter of fact, procedure PRUNE relies on the contra-positive of Claim 1: if $\varphi, \varphi\prime \in \mathcal{U}_i$ are $[E, \Delta]$-far, i.e., there exists j such that either $E^{\varphi'}_{i,j} > \Delta E^{\varphi}_{i,j}$ or $E^{\varphi'}_{i,j} < \Delta^{-1} E^{\varphi}_{i,j}$, then $E^{\varphi}_{i,j}$ and $E^{\varphi'}_{i,j}$ must be in different Δ-boxes. Accordingly, the number of Δ-boxes of the i-th task will be bounded above by the maximum number of intervals over M coordinates (which we will show is polynomial in the input size when M is constant).

We use the observations above to bound the cardinality of the pruned state space as produced by our pruning procedure. First, we need to impose a technical condition on the packing functions f_j with respect to the trimming factor Δ, so that the error propagation resulting from pruning is controlled.

Proposition 3. *If state φ is $[E, \Delta]$-close to state φ', then for any f_j, $f_k \in F_C$, $f_j(C, X, \varphi)$ is $[E, \Delta]$-close to $f_k(C, X, \varphi')$.*

Functions that satisfy the condition above will be said to have the property of being $[E, \Delta]$ *-closeness preserving*. This condition will be used later in establishing the correspondence between the state space that results from SCHEDULEEXACT and that produced by SCHEDULEPRUNED.

The pruning factor Δ: We choose the pruning factor Δ to be equal to $1 + \frac{\epsilon}{2N}$. Therefore Δ is an increasing function with respect to user specified error factor ϵ.

Algorithm 2. SCHEDULEPRUNED($\langle O_1, \ldots, O_N \rangle, \epsilon, Q$)

```
1 $\mathcal{A} \leftarrow \emptyset$
2 $\mathcal{P}_0 \leftarrow \{\langle(\emptyset, 0), \ldots, (\emptyset, 0)\rangle\}$
3 foreach C of the $\binom{\lambda + M - 1}{M}$ configurations do
4   invalidConfiguration ← true
5   for $i \leftarrow 1$ to $N$ do
6     Process $O_i$ according to $C$ to generate $X_i$
7     $\mathcal{U}_i \leftarrow \emptyset$
8     foreach $\varphi \in \mathcal{P}_{i-1}$ do
9       for $j \leftarrow 1$ to $M$ do
10        if $h_{f_j}(C, X_i, \varphi) \leqslant 0$ then
11          $\mathcal{U}_i \leftarrow \mathcal{U}_i \cup \{f_j(C, X_i, \varphi)\}$
12          invalidConfiguration ← false
13        end if
14      end for
15    end foreach
16    if invalidConfiguration is true then
17      Move to the next configuration
18    end if
19    $\mathcal{P}_i \leftarrow$ PRUNE($\mathcal{U}_i, N, \epsilon$)
20  end for
21  Examine every pair of states $\varphi$ and $\varphi\prime \in \mathcal{P}_N$. If $E(\varphi) \leqslant E(\varphi')$
    and $R(\varphi) \geqslant R(\varphi')$ then remove $\varphi'$ from $\mathcal{P}_N$
22  Sort $\mathcal{P}_N$ by non-decreasing $E(\varphi)$; sort entries with equal
    energy value by non-increasing reward $R(\varphi)$
23  $\hat{\varphi} \leftarrow$ the first $\varphi$ in the sorted $\mathcal{P}_N$ such that $R(\varphi) \geqslant Q$
24  if no $\varphi$ in $\mathcal{P}_N$ achieves $R(\varphi) \geqslant Q$ then
25    Move to the next configuration
26  end if
27  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(C, \hat{\varphi})\}$
28 end foreach
29 if all platform configurations are invalid then
30   Report "No feasible schedule exists" and terminate
31 end if
32 Examine every pair $(C, \varphi)$ and $(C\prime, \varphi\prime) \in \mathcal{A}$. If $E(\varphi) \leqslant E(\varphi')$
   and $R(\varphi) \geqslant R(\varphi')$ then remove $(C', \varphi')$ from $\mathcal{A}$
33 Sort $\mathcal{A}$ by non-decreasing $E(\varphi)$, sort entries with equal
   energy value by non-decreasing reward $R(\varphi)$
34 return the first $(C, \varphi)$ in the sorted $\mathcal{A}$
```

Procedure. Prune($\mathcal{U}, N, \epsilon$)

```
1 $\Delta \leftarrow (1 + \frac{\epsilon}{2N})$
2 $\mathcal{P} \leftarrow \emptyset$
3 Let $E_{\max} \leftarrow \max_j \max\{E^{\varphi}_j : \varphi \in \mathcal{U}\}$
4 Let $L = \lceil \log_{\Delta} \max(1, E_{\max}) \rceil$
5 Compute $\Delta^2, \ldots, \Delta^L$
6 foreach $\varphi \in \mathcal{U}$ do
7   for $j = 1$ to $M$ do
8     Consider $(Y^{\varphi}_j, E^{\varphi}_j, \ell^{\varphi}_j)$
9     if $E^{\varphi}_j < 1$ then
10      Set $\ell^{\varphi}_j \leftarrow 0$
11    else
      /* Find the interval in which $E^{\varphi}_j$ resides */
12      for $\ell' \leftarrow 1$ to $L$ do
13        if $\Delta^{\ell'-1} \leqslant E^{\varphi}_j \leqslant \Delta^{\ell'}$ then
14          Set $\ell^{\varphi}_j \leftarrow \ell'$ and move to $j+1$
15        end it
16      end for
17    end if
18  end for
19 end foreach
20 Group states in $\mathcal{U}$ for which $\ell^{\varphi}_j = \ell^{\varphi'}_j$ for every $j \in \{1, \ldots, M\}$,
   for every $\varphi, \varphi\prime \in \mathcal{U}$, in $\Delta$-boxes $\mathcal{B}_1, \ldots, \mathcal{B}_b$
21 for $k \leftarrow 1$ to $b$ do
22   if $\mathcal{B}_k \cap \mathcal{U} \neq \emptyset$ then
23     Add to $\mathcal{P}$ the state $\varphi \in \mathcal{B}_k$ for which $R(\varphi)$ is maximum
24   end if
25 end for
26 return $\mathcal{P}$
```

Lemma 1. *The number of states in each iteration after pruning, $|\mathcal{P}_i|$, is polynomial in N and $1/\epsilon$.*

Proof. The cardinality of $\mathcal{P}_i$ is bounded above by the number of Δ-boxes $\mathcal{B}_{k,i}$ induced by the $[E, \Delta]$-closeness relation on $\mathcal{U}_i$, which in turn is bounded above by the number of intervals L_i in the i-th iteration. Let b_i be the number of Δ-boxes $\mathcal{B}_{i,k}$, $k \in \{1, \ldots, b_i\}$. Further let

$$B_i = |\{k : \mathcal{B}_{i,k} \cap \mathcal{U}_i \neq \emptyset, \quad k \in \{1, \ldots, b_i\}\}|.$$

The number of Δ-boxes is at most $(1 + L_i)^M$ and therefore

$$|\mathcal{P}_i| = B_i \leqslant b_i \leqslant (1 + L_i)^M.$$

We know that

$$L_i = \lceil \log_{\Delta} \max(1, E_{i,\max}) \rceil = \left\lceil \frac{\ln \max(1, E_{i,\max})}{\ln(1 + (\epsilon/2N))} \right\rceil, \tag{13}$$

but

$$\frac{\ln \max(1, E_{i,\max})}{\ln\left(1 + (\epsilon/2N)\right)} \leqslant \frac{2N\left(1 + (\epsilon/2N)\right) \ln \max(1, E_{i,\max})}{\epsilon} \leqslant \frac{3N}{\epsilon} \ln \max(1, E_{i,\max}), \quad \text{[by inequality] (10)} \tag{14}$$

(inequality (14) is obtained using $\ln(x+1) \geqslant x/(x+1)$ when $x > -1$). Thus

$$L_i \leqslant \left\lceil 1 + \frac{3N}{\epsilon} \ln \max(1, E_{i,\max}) \right\rceil .$$

The cardinality of $\mathcal{P}_i$ is therefore

$$|\mathcal{P}_i| \leqslant (1 + L_i)^M \leqslant (1 + \left\lceil 1 + \frac{3N}{\epsilon} \ln \max(1, E_{i,\max}) \right\rceil)^M. \tag{15}$$

The latter bound is polynomial in N, $1/\epsilon$, and the number of bits required to encode the input in binary, where the number of processors M is constant. This concludes the proof. □

In order to obtain the approximation ratio of our FPTAS, we need a lemma relating the state spaces $\mathcal{U}$ and $\mathcal{P}$ of SchedulePruned to the state space $\mathcal{S}$ maintained by ScheduleExact. In fact, we need to show the effect of pruning on the value of the optimal solution produced by our algorithm. The following lemma shows how severely the optimal solution can deteriorate as a result of pruning.

Lemma 2. *If $\varphi^* \in \mathcal{S}_N$ is the state that yields the optimal solution in the exact DP formulation (Algorithm* ScheduleExact*), and $\tilde{\varphi} \in \mathcal{P}_N$ is the state returned by* SchedulePruned*, then $\tilde{\varphi}$ is at most $[E, \Delta^N]$-close to φ^*.*

Proof. State φ^* is produced by a chain of N-applications of functions $f_j \in \mathcal{F}_C$. Denote as $\varphi_i^* \in \mathcal{S}_i$ the state produced as a result of applying some $f_j \in \mathcal{F}_C$ to $\varphi_{i-1}^* \in \mathcal{S}_{i-1}$, i.e., $\varphi_i^* = f_j(C, X_i, \varphi_{i-1}^*)$ and thus $\varphi^* = \varphi_N^* = f_j(C, X_i, \varphi_{N-1}^*)$.

Consider the decomposition of φ^* into its N partial schedules $\varphi_1^*, \ldots, \varphi_N^*$. We show, by induction on i, that for $\varphi_i^* \in \mathcal{S}_i$, there exists a state $\tilde{\varphi}_i \in \mathcal{P}_i$ that is $[E, \Delta^i]$-close to φ_i^*. There is nothing to show for $i = 0$, since $\mathcal{S}_0 = \mathcal{P}_0$. Consider $\varphi_{i-1}^* \in \mathcal{S}_{i-1}$. By the induction hypothesis, there exists $\tilde{\varphi}_{i-1} \in \mathcal{P}_{i-1}$ that is $[E, \Delta^{i-1}]$-close to φ_{i-1}^*. By construction of $\mathcal{U}_i$, the set $\mathcal{U}_i$ contains the state $\hat{\varphi}_i = f_j(C, X_i, \tilde{\varphi}_{i-1})$. By construction of $\mathcal{P}_i$, there exists a state $\tilde{\varphi}_i \in \mathcal{P}_i$ that is $[E, \Delta]$-close to $\hat{\varphi}_i$. Since φ_{i-1}^* is $[E, \Delta^{i-1}]$-close to $\tilde{\varphi}_{i-1}$, the condition in Proposition 3 guarantees that $\tilde{\varphi}_i$ is $[E, \Delta^{i-1}]$-close to φ_i^*. Since φ_i^* is $[E, \Delta^{i-1}]$-close to $\tilde{\varphi}_i$ and $\tilde{\varphi}_i$ is $[E, \Delta]$-close to $\hat{\varphi}_i$, it follows by Proposition 2.2 that φ_i^* is $[E, \Delta^i]$-close to $\hat{\varphi}_i$. Applying this result for $i = N$ proves the lemma. □

Now we are ready to obtain the desired approximation factor of our FPTAS.

Lemma 3. *The energy expenditure $\widetilde{E}$ of the solution produced by* SchedulePruned *is at most $(1+\epsilon)$ the optimal solution E^*, i.e., $\widetilde{E} \leqslant (1+\epsilon)E^*$*

Proof. From Lemma 2 we have

$$\begin{aligned} E_{N,j}^{\tilde{\varphi}} &\leqslant \left(1 + \frac{\epsilon}{2N}\right)^N E_{N,j}^{\varphi^*} \\ &\leqslant (1+\epsilon) E_{N,j}^{\varphi^*} \end{aligned}$$

for every $j \in \{1, \ldots, M\}$, where we use the inequality $(1 + \frac{x}{n})^n \leqslant 1 + 2x$, $0 \leqslant x \leqslant 1$, $n \geqslant 1$, with $n = N$ and $x = \epsilon/2$. This is a reasonable approximation that can be verified by noticing that the left hand side is convex over [0, 1] and the right hand side is linear. To formally prove its validity in our setting, we know that:

$$\left(1 + \frac{\epsilon}{2N}\right)^N = e^{N \ln(1+\epsilon/2N)} \tag{16}$$

$$\leqslant e^{\epsilon/2} \tag{17}$$

$$\leqslant 1 + \epsilon/2 + (\epsilon/2)^2 \tag{18}$$

$$\leqslant 1 + \epsilon, \tag{19}$$

where (17) is obtained by $\ln(1+x) \leqslant x$, when $|x| \leqslant 1$, and inequality (19) follows from inequality (10) by the following

$$\begin{aligned} &0 < \epsilon \leqslant 1 \\ &0 < (\epsilon/2)^2 \leqslant \epsilon/2 \\ &\epsilon/2 < \epsilon/2 + (\epsilon/2)^2 \leqslant \epsilon \\ &1 + \epsilon/2 < 1 + \epsilon/2 + (\epsilon/2)^2 \leqslant 1 + \epsilon. \end{aligned}$$

Accordingly, the energy expenditure of our solution is

$$\begin{aligned} \widetilde{E} = E(\tilde{\varphi}) &= \sum_{j=1}^{M} E_{N,j}^{\tilde{\varphi}} \\ &\leqslant (1+\epsilon) \sum_{j=1}^{M} E_{N,j}^{\varphi^*} \\ &= (1+\epsilon) E(\varphi^*) = (1+\epsilon)\textsc{Opt}, \end{aligned}$$

which proves the lemma.□

Theorem 1. *Algorithm* SchedulePruned *is a fully polynomial time approximation scheme that runs in time polynomial in N, $1/\epsilon$, and the number of bits required to encode the input in binary, where the number of processors M is constant.*

Proof. Let us start by bounding the number of steps required by procedure Prune, which is the bottleneck of SchedulePruned. Setting the interval numbers for every energy value across all states and processors (line 6 to 19) requires at most a total of $M(L_i + 1)|\mathcal{U}_i|$ comparisons with the different $\Delta^{\ell'}$ values, where L_i is as defined in line 4. Grouping states into Δ-boxes (line 20) can be done as follows: Start with any vector in $\mathcal{U}_i$, create a Δ-box for it and insert it in this Δ-box. Then pick another vector and compare with the first vector: if both have exactly the same interval indexes for all M processors, then insert the vector being examined into the Δ -box where the first vector resides, otherwise create a new Δ-box for it and insert it there. Therefore, for the k-th vector in $\mathcal{U}_i$, say φ_k, we will need to compare the interval indexes of φ_k to only one vector in the thereby created Δ-boxes, and insert to the Δ-box to which it belongs, or otherwise create a new Δ-box for it. If, at the worst, when examining the k-th vector, every vector so far examined resides in its own Δ-box (i.e., all are Δ-far), then Prune will need to perform $k-1$ vector comparisons. Therefore, the maximum number of interval index comparisons performed by the grouping procedure is bounded above by

$$\sum_{k=2}^{|\mathcal{U}_i|} M(k-1) = \mathcal{O}(|\mathcal{U}_i|^2),$$

which is the worst case asymptotic time complexity of Prune. Processing the input as in line 6 can be done in time linear in the size of the specification of each task input per platform configuration. For the elimination of incomparable solution points (line 21), we will need to examine at most $\binom{|\mathcal{P}_N|}{2} = \mathcal{O}(|\mathcal{P}_N|^2)$ vectors. The solution points per $\mathcal{P}_N$ can be sorted (line 22) by an optimal sorting algorithm using $\Theta(|\mathcal{P}_N| \log |\mathcal{P}_N|)$ comparisons. Similarly for the set $\mathcal{A}$, we note that the cardinality of $\mathcal{A}$ cannot exceed the number of configurations Λ; therefore, elimination of incomparable solution points (line 21) will need to examine at most $\binom{|\Lambda|}{2} = \mathcal{O}(|\Lambda|^2)$ vectors, which is constant with respect to the size of the input (i.e., does not grow as the input size changes). Sorting $\mathcal{A}$ (line 33) requires $\Theta(|\Lambda| \log |\Lambda|)$, which, again, is a constant in the size of the input.

Since $|\mathcal{U}_i| \leqslant M|\mathcal{P}_{i-1}|$, we can write all of the bounds above in terms of $\mathcal{P}_{i-1}$ instead of $\mathcal{U}_i$, where $|\mathcal{P}_{i-1}|$ is as defined in (15). Let $\gamma_i = (1 + \lceil 1 + 3N/\epsilon \ln \max(1, E_{i,\max}) \rceil)$. Accordingly, the number of comparisons required for setting the index intervals in PRUNE as disscued above can be written as

$$M(L_i+1)|\mathcal{U}_i| \leqslant M^2\gamma_i\gamma_{i-1}^M.$$

Further, let $\widehat{E} = \max_i E_{i,\max}$. Let $\gamma = (1 + \lceil 1 + 3N/\epsilon \ln \max(1, \widehat{E})\rceil)$. If we write the bounds above in terms of $\mathcal{P}_{i-1}$ and sum them over N, then the asymptotic time complexity of Algorithm SCHEDULEPRUNED is

$$T(N,\epsilon) = \mathcal{O}(N\gamma^{2M}). \tag{20}$$

This concludes the proof. □

4.2.1. Practical considerations

The running time of algorithm SCHEDULEPRUNED is a polynomial with degree equal to twice the number of machines comprising the platform, $2M$. In addition, the hidden constants (at least $M^3\begin{pmatrix}\lambda+M-1\\ M\end{pmatrix}$) can be considerably large for large enough platforms. As a consequence, SCHEDULEPRUNED might not exhibit a fast running time for a considerably large number of processors. This makes our algorithm most suitable for environments that exhibit a static behavior in terms of task requirements and system conditions, so that the algorithm is executed in an offline fashion.

The running time of SCHEDULEPRUNED, however, can be significantly improved by using the efficient implementation of partitioned scheduling using a look-up table approach, which was devised by Chattopadhyay and Baruah [13]. In brief, their algorithm computes all possible configurations for each bin independently of the task set requirements *only once per platform* (this is the computationally intensive part, but the number of configurations is polynomially bounded in the size of the problem instance), and then uses the stored tables to assign tasks to processors in an efficient manner (this is the efficient implementation part). Doing so will allow SCHEDULEPRUNED to respond faster to changes in the platform, such as overload conditions, processor failures, etc., without the need to re-assemble the platform for every run of the algorithm.

5. Heurisitc: least average energy-to-reward first

Due to the potentially slow running-time of SCHEDULEPRUNED as discussed above, we sought a heuristic for task assignment that runs efficiently in practice, while not necessarily providing the same guarantees that SCHEDULEPRUNED provides, but that is close-enough to be used practically.

Eq. (8) suggests that the energy of a processor attains its minimum if tasks are assigned to processors such that the task with the smallest difference between the dynamic power and idle power is minimized across all processors in the platform. Our task assignment heuristic is based on this observation, but with a twist: since we wish to achieve a certain QoS score, the smaller the *ratio* between the energy of a task and its reward of execution on a processor, the less the energy this task incurs on the processor and the more the reward it achieves. We greedily assign tasks to processors such that the task with the minimum energy-to-reward ratio is assigned first. Given a platform configuration C, let d_i be the set of power difference-to-reward ratios for task T_i for each processor π'_j in C; $d_i = \{d_{i,j}\}_{j=1}^M$, where

$$d_{i,j} = \begin{cases} \dfrac{\dfrac{W_{i,j}}{P_i} - u_{i,j}p_{idle,j}}{r_{i,j}} & \text{if } T_i \text{ is defined on processor } \pi'_j \text{ in } C,\\ \infty & \text{otherwise.}\end{cases}$$

Table 3
Physical processor types specifications. $K = 4$, $\lambda = 11$.

Processor	Idle power p_{idle} (W)	Speed	Dynamic power p_{dyn} (W)
Processor$_1$	50	1	90.5
		2	95.64
Processor$_2$	20	1	86
		2	88
Processor$_3$			
		1	100
	90		
	2	130	
	3	170	
Processor$_4$		1	75
	30	2	80.5
		3	87
		4	92

The algorithm computes the set d_i for each task with respect to current platform configuration. Then for each d_i it adds the set $\{(i, d_{i,j})\}_{j=1}^M$ to a list; if $\mathcal{D}$ denotes the list of energy ratios, then

$$\mathcal{D} = \bigcup_{i=1}^{N}\{(i, d_{i,j})\}_{j=1}^{M}.$$

Afterward the algorithm sorts the list $\mathcal{D}$ by non-decreasing $d_{i,j}$ value. At this point, the algorithm iterates through the elements in the list, and for each $d_{i,j}$, if processor π'_j can accommodate T_i, i.e., it does not get overloaded when adding $u_{i,j}$ to its cumulative utilization, then T_i is assigned to π'_j and all its entries in the list (if any) are discarded. The algorithm continues in this manner until it reaches the end of the list, or the list becomes empty.

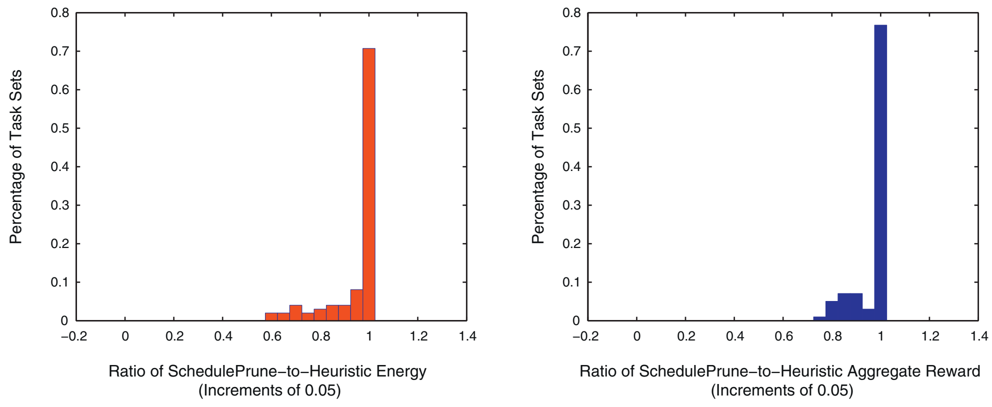
The algorithm runs in $\mathcal{O}(N^2)$ time, which is much more efficient than the running time of SCHEDULEPRUNED, shown in Eq. (20). We evaluate the efficacy of this heuristic as compared to SCHEDULEPRUNED next.

6. Simulation study

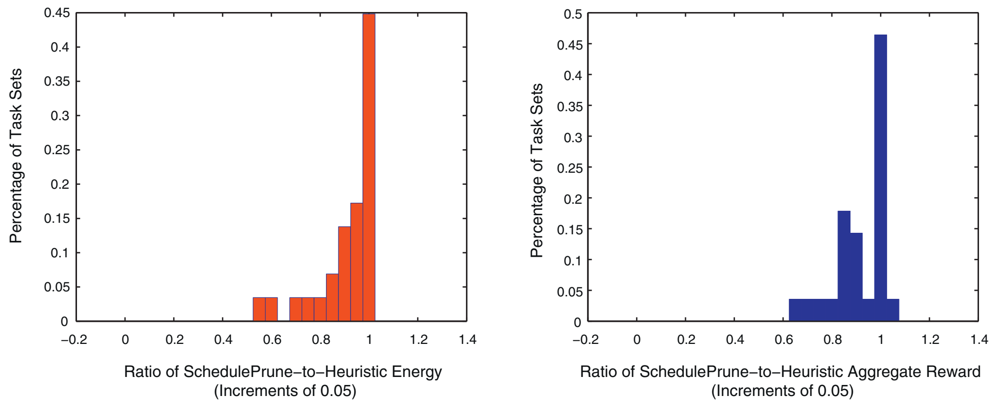
We implemented a simulation of both SCHEDULEPRUNED and the heuristic above. The purpose of this study is to compare the quality of the solutions produced by the heuristic to those of our FPTAS (SCHEDULEPRUNED), *in terms of the energy expenditure of the resulting platform and schedule*, on small to medium size platforms and task sets. We use SCHEDULEPRUNED as the baseline for the evaluation of the quality of the heuristic.

Table 3 lists the physical processor types that we will use to construct the platform. We have $K = 4$ physical processor types and $\lambda = 11$ logical processor types. We choose the energy measurement interval τ to be equal to the LCM of task periods. We model the dynamic energy component as mentioned in Section 2.3 as $W_{i,k} = u_{i,k}Lp_{dyn,k}$, where we set $\tau = L$ by our choice of the energy measurement interval.

We evaluate the algorithm for combinations of the following parameters values: $M = 2, 3$, $N = 5, 10$ and $\epsilon = 0.1, 0.9$. For each combination, we test the algorithm for 100 task sets, which we sample randomly as follows: For each task set, we draw the period P_i of each task independently and uniformly in the interval [10, 10, 000], then for each logical processor, we generate the utilization $u_{i,j}$ of task T_i as its assigned to processor π'_j independently and uniformly in the interval [0, 1]. Moreover, the reward $r_{i,j}$ that each task receives as it executes on logical processor π'_j is picked uniformly in the interval [1, 100], with a target aggregate QoS score of $Q = 300$ to be achieved.



(a) N=5. Unschedulable Schedule Pruned: (0/100), Unschedulable Heuristic: (1/100) ,Unschedulable Both:(0/100)



(b) N=5. Unschedulable Schedule Pruned: (60/100), Unschedulable Heuristic: (71/100) ,Unschedulable Both:(60/100)

Fig. 1. Histograms for the ratio of the energy expenditure of the solution produced by the heuristic to that of the solution produced by SCHEDULEPRUNED (left), and the ratio of the aggregate reward of the solutions produced by the same algorithms (right) for *100 task sets. $M=2$ machines per platform* and $\epsilon=0.1$. (a) $N=5$. Unschedulable SCHEDULEPRUNED: (0/100), Unschedulable heuristic: (1/100), Unschedulable Both: (0/100) (b) $N=10$. Unschedulable SCHEDULEPRUNED: (60/100), Unschedulable Heuristic: (71/100), Unschedulable Both: (60/100).

We show the distributions of all test task sets among both heuristic-to-SCHEDULEPRUNED energy and QoS score ratios, where we divide the ratio axis into increments of 0.05. We report the resulting histograms in Figs. 1–4.

The simulation results show that, whenever a task set is schedulable[2] by the heuristic, then it is schedulable by SCHEDULEPRUNED. Moreover, if a task set is not schedulable by SCHEDULEPRUNED, then it is not schedulable by the heuristic. This was expected since the heuristic does not consider the order of task utilizations when packing tasks into processors.

We notice a concentration of the tasks sets towards higher ratios of energies. Specifically, for the same task set, schedules generated by the heuristic consume at most twice as much energy as schedules generated by SCHEDULEPRUNED. For higher values of the error factor, the heuristic even supercedes SCHEDULEPRUNED for some task sets in terms of energy expenditure (Fig. 4(a)). Further, the QoS score achieved by the heuristic supercedes that of SCHEDULEPRUNED for most task sets, except for less than 5% of all task sets.

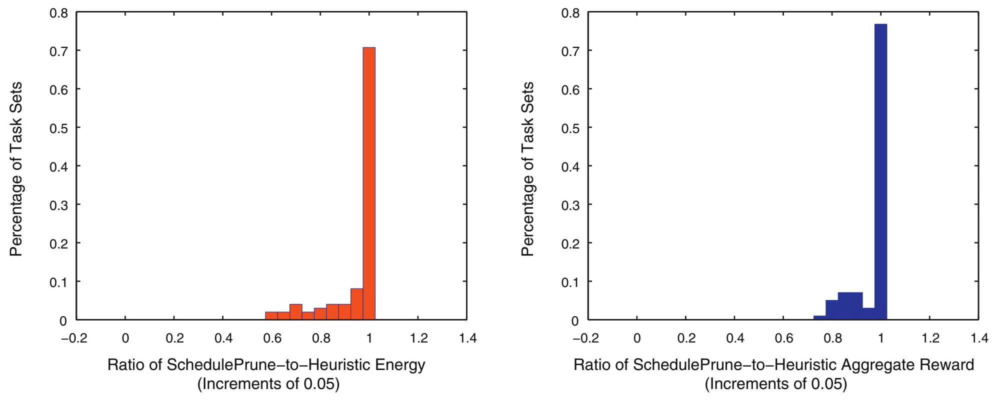
7. Related work

A vast body of research is dedicated for investigating possible techniques for saving energy on computing systems. Approaches range from designing low power hardware or that with power management capabilities to designing energy-aware scheduling algorithms.

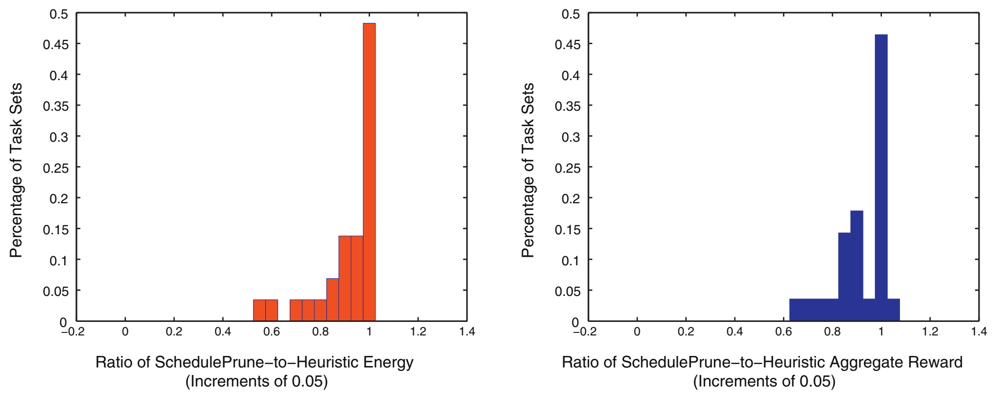
On the one hand, energy minimization can be achieved by means of *Dynamic Frequency/Voltage Scaling*, which is the most common technique used to minimize the energy consumption of processors in an online fashion. On the other hand, *Power Management* schemes aim at finding a suitable assignment of speeds to processors given prior knowledge of task parameters at design time, i.e., before tasks start execution.

In their work, Irani et al. [14] considered uniprocessor systems, and assumed that the power function $P(s)$ is continuous and convex with respect to the speed s. Further they assumed that speed is a continuous function of time $s(t)$ with no upper limit. The authors defined the *critical speed*, which is the speed at which the energy function $P(s)/s$ is at minimum. They developed an offline approximation algorithm that runs the processor at the critical speed and provided a lower bound of 3 from optimal. Moreover, the authors studied the gain of turning the processor to the dormant mode if it becomes idle, by incorporating the wakeup switching overhead to the analysis. They concluded that it is rewarding to do so if the processor's idle period is no less than the *break-even time*, which is equal to the ratio between the energy consumed in waking up from the sleep mode to the power consumption at the minimum frequency. Otherwise tasks can be run at a higher speed to create longer idle periods, where turning the processor to the dormant mode becomes beneficial. The authors also designed an online

[2] By schedulable we mean that both a feasible packing of tasks into processors exists and that the packing achieves the targets QoS score.



(a) N=5. Unschedulable Schedule Pruned: (0/100), Unschedulable Heuristic: (1/100) ,Unschedulable Both:(0/100)



(b) N=10. Unschedulable Schedule Pruned: (60/100), Unschedulable Heuristic: (71/100) ,Unschedulable Both:(60/100)

Fig. 2. Histograms for the ratio of the energy expenditure of the solution produced by the heuristic to that of the solution produced by SchedulePruned (Left), and the ratio of the aggregate reward of the solutions produced by the same algorithms (Right) for *100 task sets*. $M = 2$ *machines per platform* and $\epsilon = 0.9$. (a) $N = 5$. Unschedulable SchedulePruned: (0/100), Unschedulable Heuristic: (1/100), Unschedulable Both: (0/100) (b) $N = 10$. Unschedulable SchedulePruned: (60/100), Unschedulable Heuristic: (71/100), Unschedulable Both: (60/100).

algorithm with an optimized competitive ratio of 193 for a cubic power function. The impracticality of their solution lies in their assumptions about the continuity of speed and power.

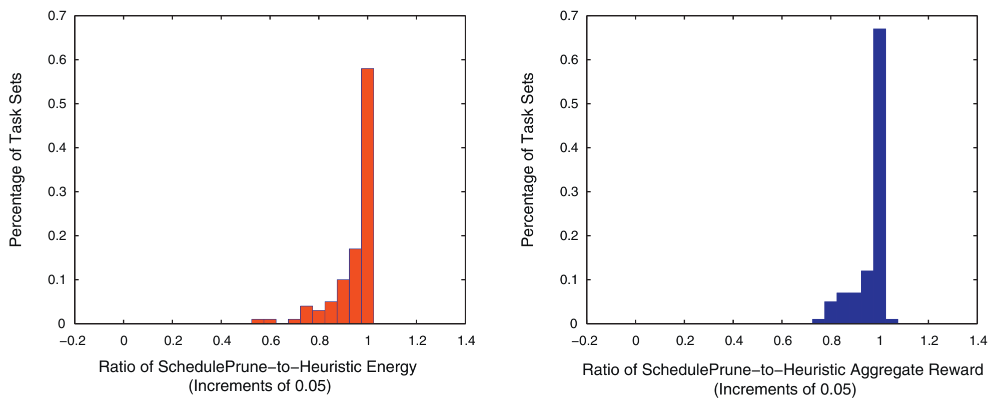
One of the significant results is that by Ishihara and Yasuura [15], where the authors considered processors with discretely variable voltages. They showed that a single voltage alternation can bring the energy expenditure to a minimum. Specifically, for an ideal system with continuous voltage levels, they found out that the voltage that brings the execution time of a task to exactly the deadline is the voltage that minimizes the energy expenditure. If that voltage is not available on the system, then its two immediate neighbors can minimize the energy consumption.

Chen [16] proposed DVS-based algorithms for minimizing the *expected* energy expenditure of frame-based tasks on uniprocessors with discrete frequencies. Their approach is probabilistic and required a discrete probability distribution of task execution cycles. The authors further considered tasks with different power characteristics. They defined an *operating point* to be the normalized energy at a certain frequency. For a set of operating points per task, the authors applied a convex hull algorithm to eliminate energy-inefficient operating points and produce a set of *usable points* that forms a convex set. Afterward, they greedily changed the operating frequency of the processor at certain points starting from the highest frequency in the ordered set of usable points. This intuitively means that the processor slows down as a task executes more.

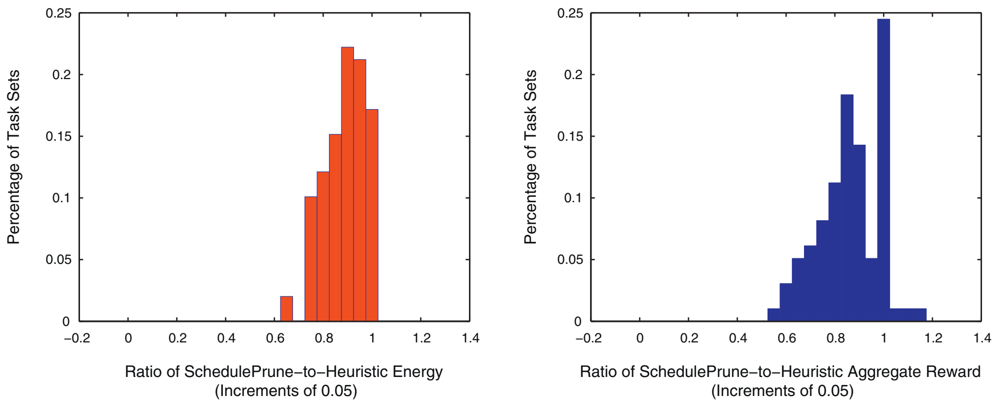
Based on the work above by Ishihara and Yasuura [15], Bini et al. [17] also incorporated the findings of Seth et al. [18] (about how tasks' WCET scale with speed) and considered task execution cycles that do not scale with speed (spent waiting for device bus). They devised algorithms for scheduling on a single processor with discrete frequency steps to minimize energy.

Most recently, Yang et al. [4] proposed an FPTAS for scheduling real time tasks on multiprocessor with discrete speeds. They used the trimming-the-state-space technique that we used in our work. The authors assumed that the WCET of tasks are given at the highest speed $s_j^{\max}$ and scaled the WCET linearly as speed changes. According to their energy model, the minimum energy is achieved by operating each processor at speed equal to $U_j s_j^{\max}$. If this speed is not available on the processor, then they chose its immediate neighbors $s_{j,k}$ and $s_{j,k+1}$ such that $s_{j,k} < U_j s_j^{\max} < s_{j,k+1}$. The energy in their case is the value of the function resulting from the linear interpolation of the points $(s_{j,k}, tP_j(s_{j,k}))$, $(s_{j,k+1}, tP_j(s_{j,k+1}))$ at $U_j s_j^{\max}$, where t is the interval of energy measurement. This assumes that energy is a linear function of the speed when the workload is fixed, falling in the subtlety that the WCEC of tasks remains constant with speed. Moreover, the authors do not consider leakage power consumption in their model.

Rusu et al. [19] considered a restricted version of the problem we study: their goal was to maximize the reward of real-time tasks executing on a uniprocessor system, subject to an energy budget



(a) N=5. Unschedulable Schedule Pruned: (0/100), Unschedulable Heuristic: (0/100) ,Unschedulable Both:(0/100)



(b) N=10. Unschedulable Schedule Pruned: (0/100), Unschedulable Heuristic: (1/100) ,Unschedulable Both:(0/100)

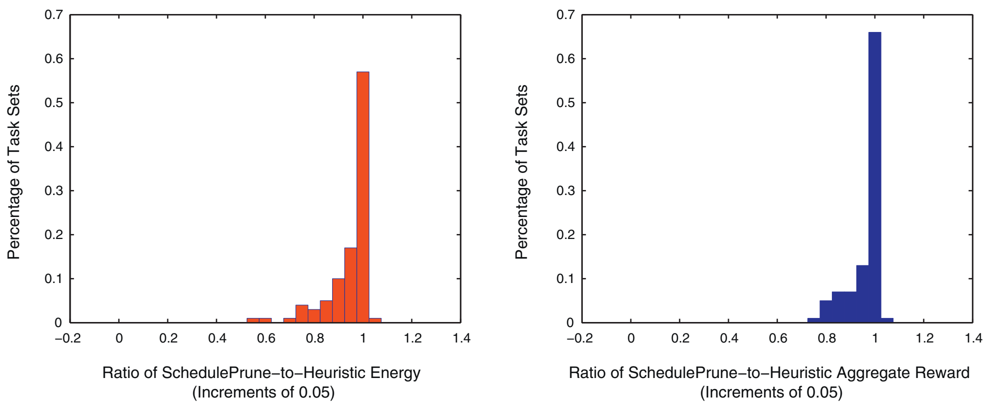
Fig. 3. Histograms for the ratio of the energy expenditure of the solution produced by the heuristic to that of the solution produced by SchedulePruned (Left), and the ratio of the aggregate reward of the solutions produced by the same algorithms (Right) for *100 task sets. $M = 3$ machines per platform* and $\epsilon = 0.1$. (a) $N = 5$. Unschedulable SchedulePruned: (0/100), Unschedulable Heuristic: (0/100), Unschedulable Both: (0/100) (b) $N = 10$. Unschedulable SchedulePruned: (0/100), Unschedulable Heuristic: (1/100), Unschedulable Both: (0/100).

that the system should never exceed. The authors, however, assume that the processor speed and the power function associated with the speed are continuous.

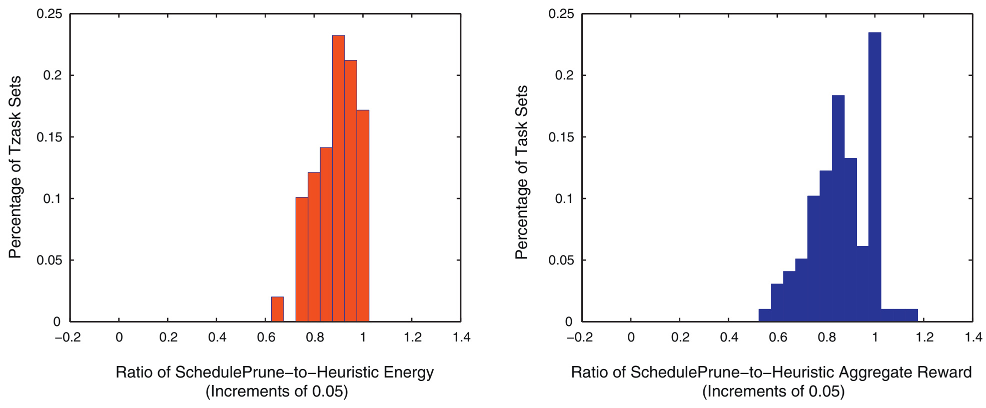
As for quality of service, the closest effort related to our work is that of Lee et al. [20] on QoS optimization with discrete QoS options. The resource allocation methods described by Lee et al. is applicable to the problem described in this article but the methods with good approximation ratios have pseudo-polynomial runtime complexity while our algorithms are polynomial-time approximations. Lee et al. [20] developed a framework for optimizing QoS with discrete settings in real-time environments from the *end users'* perspective. The authors considered QoS quantitatively and presented resource planning algorithms for multiple applications, multiple resources (disk, network, processor, etc.) and multiple QoS dimensions, so as to maximize the *utility* of end users. Their best algorithm in terms of performance is a PTAS for solving the *single resource multiple QoS dimensions* problem (we look at the case of multiple processors or resources and one QoS dimension). Their algorithm is based on dynamic programming, and produces a solution that is at most $(1 - \epsilon)$ far from optimal. Further, the authors developed a user friendly interface through which end users can specify their QoS requirements by choosing a parameterized utility curve that best matches their needs.

The *imprecise computation* model (Liu et al. [21]) divides the execution requirement of a job into *mandatory* and *optional* cycles. A task should be fully granted the execution of its mandatory requirements before it reaches its deadline, whereas the optional execution is a reward that the task receives depending on the state of the system. Accordingly, the reward is an increase in the precision of computation (e.g., increased frame rate in video streaming applications), and the goal is to maximize the reward (the amount of optional part executed), or minimize the error (the amount of unfinished optional part) in a dual sense. Khemka et al. [22] developed an optimization scheme for reducing error when imprecise tasks are scheduled on a multiprocessor system. In their model they assumed continuous job sizes and convexity conditions on the error functions, and they permitted jobs to start on one processor and then migrate to other processors. In the problems described in this article, utilization is varied in discrete steps, and each step is associated with a quality index. This approach offers some new ideas for task allocation.

The *Increasing Reward with Increasing Service* (IRIS) model (Dey et al. [23]) does not require the execution times of tasks to be known apriori. A task receives as much execution time (value) as possible before it deadline expires, with no upper limits on the obtainable reward. Both the imprecise computation and IRIS



(a) N=5. Unschedulable Schedule Pruned: (0/100), Unschedulable Heuristic: (0/100) ,Unschedulable Both:(0/100)



(b) N=10. Unschedulable Schedule Pruned: (0/100), Unschedulable Heuristic: (1/100) ,Unschedulable Both:(0/100)

Fig. 4. Histograms for the ratio of the energy expenditure of the solution produced by the heuristic to that of the solution produced by SchedulePruned (Left), and the ratio of the aggregate reward of the solutions produced by the same algorithms (Right) for *100 task sets*. $M = 3$ *machines per platform* and $\epsilon = 0.9$. (a) $N = 5$. Unschedulable SchedulePruned: (0/100), Unschedulable Heuristic: (0/100), Unschedulable Both: (0/100) (b) $N = 10$. Unschedulable SchedulePruned: (0/100), Unschedulable Heuristic: (1/100), Unschedulable Both: (0/100).

models assume that the reward function is a non-decreasing concave function.

8. Conclusions

Designing real-time computing systems to satisfy quality of service requirements while simultaneously reducing energy consumption is a computationally hard problem. While it may not be possible to obtain an optimal solution in polynomial time, we have been able to establish that near-optimal solutions can be obtained in polynomial time.

The strengths of our work are in this joint treatment of QoS and energy issues and in the accurate modeling of the execution time variations of tasks with processor speeds, and in being able to capture energy consumption in resources other than the processor. The holistic modeling of execution times and energy consumption improves the state of the art in this design problem.

We chose to restrict our attention to implicit-deadline real-time tasks and employed utilization bounds as a test for schedulability. We would like to remark that this is not a significant restriction in itself. The framework can support exact schedulability tests and arbitrary deadlines as long as one employs a polynomial time schedulability test. While exact schedulability tests do not run in polynomial time, we can instead use approximate tests (such as the one proposed by Chakraborty et al. [24]) to obtain polynomial time approximation schemes.

We have considered the situation when we have a fixed number of processor types and processors belong to one of these processor types. This opens up the way for further work on heuristics that have good performance and are faster than the dynamic programming approach that we have presented. Limited heterogeneity in compute units is a dominant choice in the design of embedded systems and our scheme is well suited to addressing this common case. Recently, Andersson et al. [25] have examined the problem of task allocation without energy or QoS considerations on a platform with two processor types. Their heuristic methods may be an initial direction to consider for the more general problem we have defined.

Whereas we have considered the quality of service as being related only to the processor type that a task is assigned to, we believe that it is possible to extend our work to cover the case when a task has multiple service classes. Consider, as an example, a video application that can operate a high, medium or low quality. This adds another choice dimension where one needs to select the appropriate service class for a task in conjunction with the choices that we have discussed in this article. Task allocation and scheduling with multiple classes of service per task is a problem that has theoretical and practical implications for the design of real-time systems.

We also note that we have dealt with the case where energy consumed by executing a job is independent of other jobs that are executed on the same processor. This may not always be the case.

For example, jobs that have mutual cache interference may lead to higher energy consumption because of an increased number of cache misses that need to be satisfied by main memory. Considering such interference between tasks is a direction for future work. It first needs extensive measurements and methods for estimating the interference between tasks.

References

[1] R. Jejurikar, C. Pereira, R. Gupta, Leakage aware dynamic voltage scaling for real-time embedded systems, 2004, pp. 275–280.
[2] W. Wolf, The future of multiprocessor systems-on-chips, in: DAC '04: Proceedings of the 41st Annual Design Automation Conference, ACM, New York, NY, USA, 2004, pp. 681–685.
[3] W. Baek, T.M. Chilimbi, Green: a framework for supporting energy-conscious programming using controlled approximation, in: Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation, PLDI '10, ACM, New York, NY, USA, 2010, pp. 198–209.
[4] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, L. Thiele, An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems, in: DATE, 2009, pp. 694–699.
[5] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, R. Brown, Mibench: a free, commercially representative embedded benchmark suite, 2001, pp. 3–14.
[6] Intel Corporation, Intel® Core^{TM}i-7 900 Mobile Processor Extreme Edition Series, Intel Core i7-800 and i7-700 Mobile Processor Series Datasheet. http://download.intel.com/design/processor/datashts/320765.pdf, 2009.
[7] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of ACM 20 (1973) 46–61.
[8] S. Baruah, R. Howell, L. Rosier, Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor, Real-Time Systems 2 (1990) 301–324.
[9] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, W. H. Freeman, 1979.
[10] G.J. Woeginger, When does a dynamic programming formulation guarantee the existence of an fptas? in: Electronic Colloquium on Computational Complexity (ECCC), 2001.
[11] O.H. Ibarra, C.E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems Journal of ACM 22 (1975) 463–468.
[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd ed., McGraw-Hill Science/Engineering/Math, 2009.
[13] B. Chattopadhyay, S. Baruah, A lookup-table driven approach to partitioned scheduling, in: Real-Time and Embedded Technology and Applications Symposium (RTAS), 17th IEEE, 2011, pp. 257–265.
[14] S. Irani, S. Shukla, R. Gupta, Algorithms for power savings, ACM Transactions on Algorithms 3 (2007) 41.
[15] T. Ishihara, H. Yasuura, Voltage scheduling problem for dynamically variable voltage processors, in: ISLPED '98: Proceedings of the 1998 International Symposium on Low Power Electronics and Design, ACM, New York, NY, USA, 1998, pp. 197–202.
[16] J.-J. Chen, Expected energy consumption minimization in dvs systems with discrete frequencies, in: SAC '08: Proceedings of the 2008 ACM Symposium on Applied Computing, ACM, New York, NY, USA, 2008, pp. 1720–1725.
[17] E. Bini, G. Buttazzo, G. Lipari, Minimizing cpu energy in real-time systems with discrete speed management, ACM Transactions on Embedded Computing Systems 8 (2009) 1–23.
[18] K. Seth, A. Anantaraman, F. Mueller, E. Rotenberg, Fast: frequency-aware static timing analysis, ACM Transactions on Embedded Computing Systems 5 (2006) 200–224.
[19] C. Rusu, R. Melhem, D. Mossé, Maximizing rewards for real-time applications with energy constraints, ACM Transactions on Embedded Computing Systems 2 (2003) 537–559.
[20] C. Lee, J. Lehoezky, R. Rajkumar, D. Siewiorek, On quality of service optimization with discrete qos options, in: Real-Time Technology and Applications Symposium, 1999. Proceedings of the Fifth IEEE, 1999, pp. 276–286.
[21] J. Liu, K.-J. Lin, W.-K. Shih, A.-s. Yu, J.-Y. Chung, W. Zhao, Algorithms for scheduling imprecise computations, Computer 24 (1991) 58–68.
[22] A. Khemka, R. K. Shyamsundar, K. V. Subrahmanyam, Multiprocessors scheduling for imprecise computations in a hard real-time environment, in: Proceedings of the International Parallel Processing Symposium, 1993, pp. 374–378.
[23] J. Dey, J. Kurose, D. Towsley, On-line scheduling policies for a class of iris (increasing reward with increasing service) real-time tasks, Computers, IEEE Transactions 45 (1996) 802–813.
[24] S. Chakraborty, S. Künzli, L. Thiele, Approximate schedulability analysis, in: RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium, IEEE Computer Society, Washington, DC, USA, 2002, p. 159.
[25] B. Andersson, G. Raravi, K. Bletsas, Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processor, in: To Appear at the 31st IEEE Real-Time Systems Symposium, November 30–December 3, San Diego, CA, USA, 2010.

Bader N. Alahmad is currently a Ph.D. student in the department of Electrical and Computer Engineering at the University of British Columbia, Canada. He received his M.A.Sc in Computer Engineering from the University of British Columbia in 2011, and a B.S in computer engineering from Jordan University of Science and Technology in 2007. His main research interests include Real-time Computing, Scheduling Theory and Combinatorial Optimization.

Sathish Gopalakrishnan is an assistant professor in the department of Electrical and Computer Engineering at the University of British Columbia, Canada. He obtained his Ph.D. in Computer Science and an MS in Applied Mathematics from the University of Illinois at Urbana-Champaign in 2004 and 2005, respectively. His research interests are distributed, embedded and real-time systems: design, analysis, implementation and evaluation of distributed systems, with an emphasis on timeliness, energy efficiency and reliability.