Real-Time Syst
DOI 10.1007/s11241-017-9291-6

Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors

Mason Thammawichai[1] · **Eric C. Kerrigan**[2]

Abstract We propose three novel mathematical optimization formulations that solve the same two-type heterogeneous multiprocessor scheduling problem for a real-time taskset with hard constraints. Our formulations are based on a global scheduling scheme and a fluid model. The first formulation is a mixed-integer nonlinear program, since the scheduling problem is intuitively considered as an assignment problem. However, by changing the scheduling problem to first determine a task workload partition and then to find the execution order of all tasks, the computation time can be significantly reduced. Specifically, the workload partitioning problem can be formulated as a continuous nonlinear program for a system with continuous operating frequency, and as a continuous linear program for a practical system with a discrete speed level set. The latter problem can therefore be solved by an interior point method to any accuracy in polynomial time. The task ordering problem can be solved by an algorithm with a complexity that is linear in the total number of tasks. The work is evaluated against existing global energy/feasibility optimal workload allocation formulations. The results illustrate that our algorithms are both feasibility optimal and energy optimal for both implicit and constrained deadline tasksets. Specifically, our algorithm can achieve up to 40% energy saving for some simulated tasksets with constrained deadlines. The benefit of our formulation compared with existing work is that our algorithms can solve a more general class of scheduling problems due to incorporat-

✉ Mason Thammawichai
m.thammawichai12@imperial.ac.uk

Eric C. Kerrigan
e.kerrigan@imperial.ac.uk

[1] The Department of Aeronautics, Imperial College London, London SW7 2AZ, UK

[2] The Department of Electrical & Electronic Engineering and The Department of Aeronautics, Imperial College London, London SW7 2AZ, UK

Published online: 01 September 2017

ing a scheduling dynamic model in the formulations and allowing for a time-varying speed profile.

Keywords Real-time systems · Power-aware computing · Optimal scheduling · Dynamic voltage scaling · Optimal control

1 Introduction

Efficient energy management has become an important issue for modern computing systems due to higher computational power demands in today's computing systems, e.g. sensor networks, satellites, multi-robot systems, as well as personal electronic devices. There are two common schemes used in modern computing energy management systems. One is dynamic power management (DPM), where certain parts of the system are turned off during the processor idle state. The other is dynamic voltage and frequency scaling (DVFS), which reduces the energy consumption by exploiting the relation between the supply voltage and power consumption. In this work, we consider the problem of scheduling real-time tasks on heterogeneous multiprocessors under a DVFS scheme. The objective is to minimize energy consumption, while ensuring that both the execution cycle requirement and timeliness constraints of real-time tasks are satisfied. This problem is known to be NP-Complete, in general (Ullman 1975). However, one of our algorithms involves solving an LP, for which a solution can be computed to an arbitrary accuracy in polynomial time using a suitably-chosen interior point method (Boyd and Vandenberghe 2004).

1.1 Terminologies and definitions

This section provides basic terminologies and definitions used throughout the paper.

Task T_i: An aperiodic task T_i is defined as a triple $T_i := (c_i, d_i, b_i)$; c_i is the required number of CPU cycles needed to complete the task, d_i is the task's relative deadline and b_i is the arrival time of the task. A periodic task T_i is defined as a triple $T_i := (c_i, d_i, p_i)$ where p_i is the task's period. If the task's deadline is equal to its period, the task is said to have an 'implicit deadline'. The task is considered to have a 'constrained deadline' if its deadline is not larger than its period, i.e. $d_i \leq p_i$. In the case that the task's deadline can be less than, equal to, or greater than its period, it is said to have an 'arbitrary deadline'. Throughout the paper, we will refer to a task as an aperiodic task model unless stated otherwise, because a periodic task can be transformed into a collection of aperiodic tasks with appropriately defined arrival times and deadlines, i.e. the jth instance of a periodic task T_i, where $j \geq 1$, arrives at time $(j-1)p_i$, has the required execution cycles c_i and an absolute deadline at time $(j-1)p_i + d_i$. Moreover, for a periodic taskset, we only need to find a valid schedule within its hyperperiod $\mathcal{L}$, defined as the least common multiple (LCM) of all task periods, i.e. the total number of job instances of a periodic task T_i during the hyperperiod $\mathcal{L}$ is equal to $\mathcal{L}/p_i$. The taskset is defined as a set of all tasks. The taskset is feasible if there exists a schedule such that no task in the taskset misses the deadline.

Speed s^r: The operating speed s^r is defined as the ratio between the operating frequency f^r of processor type-r and the maximum system frequency f_{max}, i.e. $s^r := f^r/f_{max}$, $f_{max} := \max\{\max\{f^r \mid r \in R\}\}$, where $R := \{1, 2, \ldots, \kappa\}$ and κ is the total number of processor types. Note that the superscript r is used to denote the type-r processor.

Minimum execution time[1] $\underline{x}_i^r$: The minimum execution time $\underline{x}_i^r$ is the execution time of task T_i when executed at the maximum system frequency f_{max}, i.e. $\underline{x}_i^r := c_i^r/f_{max}$, where c_i^r is the CPU cycles required to complete a task T_i when executed on a processor of type r.

Task density[2] $\delta_i(s_i)$: For a periodic task, a task density $\delta_i(s_i)$ is defined as the ratio between the task execution time and the minimum of its deadline and its period, i.e. $\delta_i(s_i) := c_i/(s_i f_{max} \min\{d_i, p_i\})$, where s_i is the task execution speed.

Taskset density $D(s_i)$: A taskset density $D(s_i)$ of a periodic taskset is defined as the summation of all task densities in the taskset, i.e. $D(s_i) := \sum_{i=1}^{n} \delta_i(s_i)$. The minimum taskset density D is given by $D := \sum_{i=1}^{n} \delta_i(1)$.

System capacity C: The system capacity C is defined as $C := \sum_r s_{max}^r m_r$, where s_{max}^r is the maximum speed of processor type-r, i.e. $s_{max}^r := f_{max}^r/f_{max}$, $f_{max}^r := \max f^r$, m_r is the total number of processors of type-r.

Migration scheme: A global scheduling scheme allows task migration between processors and a partitioned scheduling scheme does not allow task migration.

Feasibility optimal: An algorithm is feasibility optimal if the algorithm is guaranteed to be able to construct a valid schedule such that no deadlines are missed, provided a schedule exists.

Energy optimal: An algorithm is energy optimal when it is guaranteed to find a schedule that minimizes the energy, while meeting the deadlines, provided such a schedule exists.

Step function: A function $f : X \to \mathbb{R}$ is a step (also called a piecewise constant) function, denoted $f \in \mathcal{PC}$, if there exists a finite partition $\{X_1, \ldots, X_p\}$ of $X \subseteq \mathbb{R}$ and a set of real numbers $\{\phi_1, \ldots, \phi_p\}$ such that $f(x) = \phi_i$ for all $x \in X_i, i \in \{1, \ldots, p\}$.

1.2 Related work

Due to the heterogeneity of the processors, one should not only consider the different operating frequency sets among processors, but also the hardware architecture of the processors, since task execution time will be different for each processor type. In other words, the system has to be captured by two aspects: the difference in operating speed sets and the execution cycles required by different tasks on different processor types (Fig. 1).

[1] In the literature, this is often called 'best-case execution time'. However, in the case where the speed is allowed to vary, using the term 'minimum execution time' makes more sense, since the execution time increases as the speed is scaled down. For simplicity of exposition, we also assume no uncertainty, hence 'best-case' is not applicable here. Extensions to uncertainty should be relatively straightforward, in which case $\underline{x}_i$ then becomes 'minimum best-case execution time'.

[2] When all tasks are assumed to have implicit deadlines, this is often called 'task utilization'.

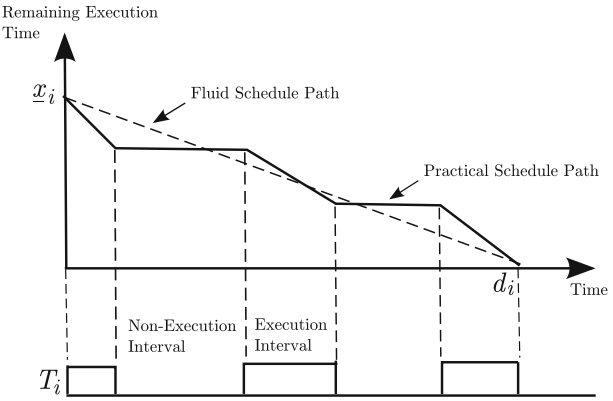


Fig. 1 Fluid schedule and a practical schedule (Cho et al. 2006)

With these aspects, fully-migration/global based scheduling algorithms, where tasks are allowed to migrate between different processor types, are not suitable in practice, since it will be difficult to identify how much computational work is executed on one processor type compared to another processor type due to differences in instruction sets, register formats, etc. Thus, most of the work related to heterogeneous multiprocessor scheduling are partition-based/non-preemptive task scheduling algorithms (Yu and Prasanna 2002; Leung et al. 2004; Yang et al. 2009; Goh et al. 2009; Chen et al. 2009; Li and Wu 2012; Awan and Petters 2013), i.e. tasks are partitioned onto one of the processor types and a well-known uniprocessor scheduling algorithm, such as Earliest Deadline First (EDF) (Liu and Layland 1973), is used to find a valid schedule. With this scheme, the heterogeneous multiprocessor scheduling problem is reduced to a task partitioning problem, which can be formulated as an integer linear program (ILP). Examples of such work are Yu and Prasanna (2002) and Chen et al. (2009).

However, with the advent of ARM two-type heterogeneous multicores architecture, such as the big.LITTLE architecture (ARM 2013), that supports task migrations among different core types, a global scheduling algorithm is possible. In Chwa et al. (2014, 2015), the first energy-aware global scheduling framework for this special architecture is presented, where an algorithm called Hetero-Split is proposed to solve a workload assignment and a Hetero-Wrap algorithm to solve a schedule generation problem. Their framework is similar to ours, except that we adopt a fluid model to represent a scheduling dynamic, our assigned operating frequency is time-varying and the CPU idle energy consumption is also considered.

A fluid model is the ideal schedule path of a real-time task. The remaining execution time is represented by a straight line where the magnitude of the slope of the line is the task execution speed. However, a practical task execution path is nonlinear, since a task may be preempted by other tasks. The execution interval of a task is represented by a line with a negative slope and a non-execution interval is represented by a line with zero slope.

There are at least two well-known homogeneous multiprocessor scheduling algorithms that are based on a fluid scheduling model: Proportionate-fair (Pfair) (Baruah

et al. 1993) and Largest Local Remaining Execution Time First (LLREF) (Cho et al. 2006). Both Pfair and LLREF are global scheduling algorithms. By introducing the notion of *fairness*, Pfair ensures that at any instant no task is one or more quanta (time intervals) away from the task's fluid path. However, the Pfair algorithm suffers from a significant run-time overhead, because tasks are split into several segments, incurring frequent algorithm invocations and task migrations. To overcome the disadvantages of quantum-based scheduling algorithms, the LLREF algorithm splits/preempts a task at two scheduling events within each time interval (Cho et al. 2006). One occurs when the remaining time of an executing task is zero and it is better to select another task to run. The other event happens when the task has no laxity, i.e. the difference between the task deadline and the remaining execution time left is zero, hence the task needs to be selected immediately in order to finish the remaining workload in time.

The unified theory of the deadline partitioning technique and its feasibility optimal versions, called DP-FAIR, for periodic and sporadic tasks are given in Levin et al. (2010). Deadline Partitioning (DP) (Levin et al. 2010) is the technique that partitions time into intervals bounded by two successive task deadlines, after which each task is allocated the workload and is scheduled at each time interval. A simple optimal scheduling algorithm based on DP-FAIR, called DP-WRAP, was presented in Levin et al. (2010). The DP-WRAP algorithm partitioned time according to the DP technique and, at each time interval, the tasks are scheduled using McNaughton's wrap around algorithm (McNaughton 1959). McNaughton's wrap around algorithm aligns all task workloads along a real number line, starting at zero, then splits tasks into chunks of length 1 and assigns each chunk to the same processor. Note that the tasks that have been split migrate between the two assigned processors. The work of Levin et al. (2010) was extended in Funk et al. (2012) and Wu et al. (2012) by incorporating a DVFS scheme to reduce power consumption.

However, the algorithms that are based on the fairness notion (Cho et al. 2006; Funaoka et al. 2008a, b; Levin et al. 2010; Funk et al. 2012; Wu et al. 2012) are feasibility optimal, but have hardly been applied in a real system, since they suffer from high scheduling overheads, i.e. task preemptions and migrations. Recently, two feasibility optimal algorithms that are not based on the notion of fairness have been proposed. One is the RUN algorithm (Regnier et al. 2011), which uses a dualization technique to reduce the multiprocessor scheduling problem to a series of uniprocessor scheduling problems. The other is U-EDF (Nelissen et al. 2012), which generalises the earliest deadline first (EDF) algorithm to multiprocessors by reducing the problem to EDF on a uniprocessor.

Alternatively to the above methods, the multiprocessor scheduling problem can also be formulated as an optimization problem. However, since the problem is NP-hard (Lawler 1983), in general, an approximated polynomial-time heuristic method is often used. An example of these approaches can be found in Chen et al. (2008) and Xian et al. (2007), which consider energy-aware multiprocessor scheduling with probabilistic task execution times. The tasks are partitioned among the set of processors, followed with computing the running frequency based on the task execution time probabilities. Among all of the feasibility assignments, an optimal energy consumption assignment is chosen by solving a mathematical optimization problem, where the objective is to minimize some energy function. The constraints are to ensure that

all tasks will meet their deadlines and only one processor is assigned to a task. In partitioned scheduling algorithms, such as Chen et al. (2008) and Xian et al. (2007), once a task is assigned to a specific processor, the multiprocessor scheduling problem is reduced to a set of uniprocessor scheduling problems, which is well studied (Chen and Kuo 2007). However, a partitioned scheduling method cannot provide a feasibility optimal schedule.

For this work, we are interested in solving the scheduling problem of a two-type heterogeneous multiprocessor system, e.g. the ARM big.LITTLE architecture. Specifically, we formulate a real-time multiprocessor scheduling problem as an infinite-dimensional continuous-time optimal control problem. Three mathematical programming formulations to solve a hard real-time task scheduling problem on heterogeneous multiprocessor systems with DVFS capabilities are proposed. First is a mixed-integer nonlinear program (MINLP), which adopts the fluid model used in Baruah et al. (1993), Cho et al. (2006), Funaoka et al. (2008a, b) to represent a scheduling dynamic. The MINLP formulation relies on the optimal control of a given system's dynamic to globally solve for a valid schedule, while solutions are obtained by solving each instance of the time interval obtained using the DP technique (Cho et al. 2006; Funaoka et al. 2008a, b; Levin et al. 2010; Funk et al. 2012; Wu et al. 2012). By determining a percentage of a task's execution time, rather than task assignments, the same scheduling problem can be divided into two sub-problems, i.e. a workload partitioning problem and a task ordering problem. A nonlinear program (NLP) and a linear program (LP) are proposed to solve the workload partitioning problem for a system with continuous operating speed and a practical system where a processor has a discrete operating speed set, respectively.

1.3 Contribution

The main contributions of this work are:

- The scheduling problem is decoupled into two sub-problems and the optimal control is adopted to solve the workload partitioning problem.
- We provide a generalised optimal speed profile solution to a uniprocessor scheduling problem with real-time taskset. That is, (i) for any given workload within the time interval, the optimal speed profile composes of at most two speed levels. (ii) For a processor with a convex power consumption model, the optimal speed profile is a constant. (iii) In general, the time-varying speed profile is better than a constant speed profile if the power function is not convex.
- The first multiprocessor scheduling algorithm that is both feasibility optimal and energy optimal.
- Our formulations are capable of solving a multiprocessor scheduling problem with any periodic tasksets as well as aperiodic tasksets, compared to existing work, due to the incorporation of a scheduling dynamic and a time-varying speed profile.
- Moreover, the proposed formulations can also be extended to a multicore architecture, which only allows frequency to be changed at a cluster-level, rather than at a core-level, as explained in Sect. 2.3.

1.4 Outline

This paper is organized as follows: Sect. 2 defines our feasibility scheduling problem in detail. Details on solving the scheduling problem with finite-dimensional mathematical optimization is given in Sect. 3. The optimality problem formulations are presented in Sect. 4. The simulation setup and results are presented in Sect. 5. Finally, conclusions and future work are discussed in Sect. 6.

2 Feasibility problem formulation

Though our objective is to minimize the total energy consumption, we will first consider a feasiblity problem before presenting an optimality problem.

2.1 System model

We consider a set of n real-time tasks that are to be partitioned on a two-type heterogeneous multiprocessor system composed of m_r processors of type-r, $r \in R$. We will assume that the system supports task migration among processor types, e.g. sharing the same instruction set and having a special interconnection for data transfer between processor types. Note that c_i is the same for all processor types, since the instruction set is the same.

2.2 Task/processor assumptions

All tasks do not share resources, do not have any precedence constraints and are ready to start at the beginning of the execution. A task can be preempted/migrated between different processor types at any time. The cost of preemption and migration is assumed to be negligible or included in the minimum task execution times. Processors of the same type are homogeneous, i.e. having the same set of operating frequencies and power consumptions. Each processor's voltage/speed can be adjusted individually. Additionally, for an ideal system, a processor is assumed to have a continuous speed range. For a practical system, a processor is assumed to have a finite set of operating speed levels.

2.3 Scheduling as an optimal control problem

Below, we will refer to the sets $I := \{1, \ldots, n\}$, $K^r := \{1, \ldots, m_r\}$ and $\Gamma := [0, L]$, where L is the largest deadline of all tasks. Note that $\forall i, \forall k, \forall r, \forall t$ are shorthand notations for $\forall i \in I, \forall k \in K^r, \forall r \in R, \forall t \in \Gamma$, respectively. The scheduling problem can therefore be formulated as the following infinite-dimensional continous-time control problem:

$$
\begin{aligned}
&\text{find} \quad x_i(\cdot), a_{ik}^r(\cdot), s_k^r(\cdot), \ \forall i \in I, k \in K^r, r \in R \\
&\text{subject to} \\
&x_i(b_i) = \underline{x}_i, && \forall i && \text{(1a)} \\
&x_i(t) = 0, && \forall i, t \notin [b_i, b_i + d_i) && \text{(1b)} \\
&\dot{x}_i(t) \geq -\sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} a_{ik}^r(t) s_k^r(t), && \forall i, t, \quad \text{a.e.} && \text{(1c)} \\
&\sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} a_{ik}^r(t) \leq 1, && \forall i, t && \text{(1d)} \\
&\sum_{i=1}^{n} a_{ik}^r(t) \leq 1, && \forall k, r, t && \text{(1e)} \\
&s_k^r(t) \in S^r, && \forall k, r, t && \text{(1f)} \\
&a_{ik}^r(t) \in \{0, 1\}, && \forall i, k, r, t && \text{(1g)} \\
&a_{ik}^r(\cdot) \in \mathcal{PC}, s_k^r(\cdot) \in \mathcal{PC}, && \forall i, k, r, t && \text{(1h)}
\end{aligned}
$$

where the state $x_i(t)$ is the remaining minimum execution time of task T_i at time t, the control input $s_k^r(t)$ is the execution speed of the kth processor of type-r at time t and the control input $a_{ik}^r(t)$ is used to indicate the processor assignment of task T_i at time t, i.e. $a_{ik}^r(t) = 1$ if and only if task T_i is active on processor k of type-r. Notice that here we formulated the problem with speed selection at a core-level; a stricter assumption of a multicore architecture, i.e. a cluster-level speed assignment, is straightforward. Particularly, by replacing a core-level speed assignment s_k^r with a cluster-level speed assignment s^r in the above formulation.

The initial conditions on the minimum execution time of all tasks and task deadline constraints are specified in (1a) and (1b), respectively. The fluid model of the scheduling dynamic is given by the differential constraint (1c). Constraint (1d) ensures that each task will be assigned to at most one non-idle processor at a time. Constraint (1e) quarantees that each non-idle processor will only be assigned to at most one task at a time. The speeds are constrained by (1f) to take on values from $S^r \subseteq [0, 1]$. Constraint (1g) emphasise that task assignment variables are binary. Lastly, (1h) denotes that the control inputs should be step functions.

Fact 1 *A solution to* (1) *where* (1c) *is satisfied with equality can be constructed from a solution to* (1)*.*

Proof Let (a, s, x) be a feasible point to (1). Let $t_i := \min\{t \in [b_i, b_i + d_i] \mid x_i(t) \leq 0\}$, $\forall i$. Choose $(\tilde{a}, \tilde{s}, \tilde{x})$ such that (i) $\tilde{a}_{ik}^r(t)\tilde{s}_k^r(t) = a_{ik}^r(t)s_k^r(t)$, $\forall i, k, r$, $t \leq t_i$ and (ii) $\tilde{a}_{ik}^r(t)\tilde{s}_k^r(t) = 0$, $\forall i, k, r, t > t_i$. Choose $\tilde{x}_i(0) = \underline{x}_i$, $\forall i$ and $\dot{\tilde{x}}_i(t) = -\sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} \tilde{a}_{ik}^r(t)\tilde{s}_k^r(t)$, $\forall i, k, r, t$. It follows that $(\tilde{a}, \tilde{s}, \tilde{x})$ is a solution to (1) where (1c) is an equality. □

3 Solving the scheduling problem with finite-dimensional mathematical optimization

The original problem (1) will be discretized by introducing piecewise constant constraints on the control inputs s and a. Let $\mathcal{T} := \{\tau_0, \tau_1, \ldots, \tau_N\}$, which we will refer to as the major grid, denote the set of discretization time steps corresponding to the distinct arrival times and deadlines of all tasks within L, where $0 = \tau_0 < \tau_1 < \tau_2 < \cdots < \tau_N = L$.

3.1 Mixed-integer nonlinear program (MINLP-DVFS)

The above scheduling problem, subject to piecewise constant constraints on the control inputs, can be naturally formulated as an MINLP, defined below. Since the context switches due to task preemption and migration can jeopardize the performance, a variable discretization time step (Gerdts 2006) method is applied on a minor grid, so that the solution to our scheduling problem does not depend on the size of the discretization time step. Let $\{\tau_{\mu,0}, \ldots, \tau_{\mu,M}\}$ denote the set of discretization time steps on a minor grid on the interval $[\tau_\mu, \tau_{\mu+1}]$ with $\tau_\mu = \tau_{\mu,0} \leq \ldots \leq \tau_{\mu,M} = \tau_{\mu+1}$, so that $\{\tau_{\mu,1}, \ldots, \tau_{\mu,M-1}\}$ is to be determined for all μ from solving an appropriately-defined optimization problem.

Let $\forall\mu$ and $\forall\nu$ be short notations for $\forall\mu \in U := \{0, 1, \ldots, N-1\}$ and $\forall\nu \in V := \{0, 1, \ldots, M-1\}$. Define the notation $[\mu, \nu] := (\tau_{\mu,\nu}), \forall\mu, \nu$. Denote the discretized state and input sequences as

$$x_i[\mu, \nu] := x_i(\tau_{\mu,\nu}), \qquad \forall i, \mu, \nu \tag{2a}$$
$$s_k^r[\mu, \nu] := s_k^r(\tau_{\mu,\nu}), \qquad \forall k, r, \mu, \nu \tag{2b}$$
$$a_{ik}^r[\mu, \nu] := a_{ik}^r(\tau_{\mu,\nu}), \qquad \forall i, k, r, \mu, \nu \tag{2c}$$

Let $s_k^r(\cdot)$ and $a_{ik}^r(\cdot)$ be step functions inbetween time instances on a minor grid, i.e.

$$s_k^r(t) = s_k^r[\mu, \nu], \qquad \forall t \in [\tau_{\mu,\nu}, \tau_{\mu,\nu+1}), \mu, \nu \tag{3a}$$
$$a_{ik}^r(t) = a_{ik}^r[\mu, \nu], \qquad \forall t \in [\tau_{\mu,\nu}, \tau_{\mu,\nu+1}), \mu, \nu \tag{3b}$$

Let Λ denote the set of all tasks within L, i.e. $\Lambda := \{T_i \mid i \in I\}$. Define a task arrival time mapping $\Phi_b : \Lambda \to U$ by $\Phi_b(T_i) := \mu$ such that $\tau_\mu = b_i$ for all $T_i \in \Lambda$ and a task deadline mapping $\Phi_d : \Lambda \to U \cup \{N\}$ by $\Phi_d(T_i) := \mu$ such that $\tau_\mu = b_i + d_i$ for all $T_i \in \Lambda$. Define $\mathcal{U}_i := \{\mu \in U \mid \Phi_b(T_i) \leq \mu < \Phi_d(T_i)\}, \ \forall i \in I$ and let $\forall\mu_i$ be short notation for $\forall\mu \in \mathcal{U}_i$.

By solving a first-order ODE with piecewise constant input, a solution of the scheduling dynamic (1c) has to satisfy the difference constraint

$$x_i[\mu, \nu+1] \geq x_i[\mu, \nu] - h[\mu, \nu] \sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} s_k^r[\mu, \nu] a_{ik}^r[\mu, \nu], \forall i, \mu_i, \nu. \tag{4a}$$

where $h[\mu, \nu] := \tau_{\mu,\nu+1} - \tau_{\mu,\nu}, \forall\mu, \nu$.

The discretization of the original problem (1) subject to piecewise constant constraints on the inputs (3) is therefore equivalent to the following finite-dimensional MINLP:

$$\begin{aligned}
&\text{find} \quad x_i[\cdot], a_{ik}^r[\cdot], s_k^r[\cdot], h[\cdot], \ \forall i \in I, k \in K^r, r \in R \\
&\text{subject to (4a) and} \\
&x_i[\Phi_b(T_i), 0] = \underline{x}_i, && \forall i && \text{(4b)} \\
&x_i[\mu, \nu] = 0, && \forall i, \mu \notin \mathcal{U}_i, \nu && \text{(4c)} \\
&\sum_{r=1}^{\kappa} \sum_{k=1}^{m_r} a_{ik}^r[\mu, \nu] \leq 1, && \forall i, \mu, \nu && \text{(4d)} \\
&\sum_{i=1}^{n} a_{ik}^r[\mu, \nu] \leq 1, && \forall k, r, \mu, \nu && \text{(4e)} \\
&s_k^r[\mu, \nu] \in S^r, && \forall k, r, \mu, \nu && \text{(4f)} \\
&a_{ik}^r[\mu, \nu] \in \{0, 1\}, && \forall i, k, r, \mu, \nu && \text{(4g)} \\
&0 \leq h[\mu, \nu], && \forall \mu, \nu && \text{(4h)} \\
&\sum_{\nu=0}^{M-1} h[\mu, \nu] \leq \tau_{\mu+1} - \tau_{\mu}, && \forall \mu && \text{(4i)}
\end{aligned}$$

where (4h)–(4i) enforce upper and lower bounds on discretization time steps.

Theorem 2 *Let the size of the minor grid* $M \geq \max_r \{m_r\}$. *A solution to* (1) *exists if and only if a solution to* (4) *exists.*

Proof Follows from the fact that if a solution exists to (1), then the Hetero-Wrap scheduling algorithm (Chwa et al. 2015) can find a valid schedule with at most $m_r - 1$ migrations within the cluster. (Chwa et al. 2015, Lemma 2).

Next, we will show that $(\tilde{a}[\cdot], \tilde{s}[\cdot], \tilde{x}[\cdot], \tilde{h}[\cdot])$, a solution to (4), can be constructed from $(a(\cdot), s(\cdot), x(\cdot))$, a solution to (1). Specifically, choose $\tilde{h}[\mu, \nu] = \tau_{\mu,\nu+1} - \tau_{\mu,\nu}$ as above and $\tilde{a}_{ik}^r[\mu, \nu] = a_{ik}^r(t)$ such that $a_{ik}^r(t)$ is a constant for all $t \in \tilde{h}[\mu, \nu]$ and

$$\tilde{h}[\mu, \nu]\tilde{a}_{ik}^r[\mu, \nu] = \int_{\tau_{\mu,\nu}}^{\tau_{\mu,\nu+1}} a_{ik}^r(t)dt, \ \forall i, r, \mu, \nu. \tag{5}$$

Then (4a)–(4c) are satisfied with $\tilde{x}_i[\mu, \nu] = x_i(\tau_{\mu,\nu}), \ \forall i, \mu, \nu$. It follows from (1d), (1e) and (1g) that (4d), (4e) and (4g) are satified, respectively. (4f) is satified with $\tilde{s}_k^r[\mu, \nu] = s_k^r(\tau_{\mu,\nu})$.

Suppose now we have $(\tilde{a}[\cdot], \tilde{s}[\cdot], \tilde{x}[\cdot], \tilde{h}[\cdot])$, a solution to (4). We can choose $(a(\cdot), s(\cdot), x(\cdot))$ to be a solution to (1) if the inputs are the step functions $a_{ik}^r(t) = \tilde{a}_{ik}^r[\mu, \nu]$ and $s_k^r(t) = \tilde{s}_k^r[\mu, \nu]$ when $\tilde{h}[\mu, \nu] \leq t - \tau_{\mu,\nu} < \tilde{h}[\mu, \nu + 1], \ \forall i, k, r, \mu, \nu$. It is simple to verify that (1) is satisfied by the above choice. □

3.2 Computationally tractable multiprocessor scheduling algorithms

The time to compute a solution to problem (4) is impractical even with a small problem size. However, if we relax the binary constraints in (4g) so that the value of a can be interpreted as the percentage of a time interval during which the task is executed (this will be denoted as ω in later formulations), rather than the processor assignment, the problem can be reformulated as an NLP for a system with continuous operating speed and an LP for a system with discrete speed levels. The NLP and LP can be solved at a fraction of the time taken to solve the MINLP above. Particularly, the heterogeneous multiprocessor scheduling problem can be simplified into two steps:

STEP 1: *Workload Partitioning* Determine the percentage of task execution times and execution speed within a time interval such that the feasibility constraints are satisfied.

STEP 2: *Task Ordering* From the solution given in the workload partitioning step, find the execution order of all tasks within a time interval such that no task will be executed on more than one processor at a time.

3.2.1 Solving the workload partitioning problem as a continuous nonlinear program (NLP-DVFS)

Since knowing the processor on which a task will be executed does not help in finding the task execution order, the corresponding processor assignment subscript k of the control variables ω and s is dropped to reduce the number of decision variables. Moreover, partitioning time using only a major grid (i.e. $M = 1$) is enough to guarantee a valid solution, i.e. the percentage of the task exection time within a major grid is equal to the sum of all percentages of task execution times in a minor grid. Since we only need a major grid, we define the notation $[\mu] := \tau_\mu$ and $h[\mu] := \tau_{\mu+1} - \tau_\mu$. Note that we make an assumption that $h[\mu] > 0,\ \forall \mu$. We also assume that the set of allowable speed levels S^r is a closed interval given by the lower bound s^r_{min} and upper bound s^r_{max}.

Consider now the following finite-dimensional NLP:

$$
\begin{aligned}
&\text{find} \quad x_i[\cdot], \omega_i^r[\cdot], s_i^r[\cdot],\ \forall i \in I, r \in R \\
&\text{subject to} \\
&\qquad x_i[\Phi_b(T_i)] = \underline{x}_i, && \forall i && \text{(6a)} \\
&\qquad x_i[\mu] = 0, && \forall i, \mu \notin \mathcal{U}_i && \text{(6b)} \\
&\qquad x_i[\mu+1] \geq x_i[\mu, \nu] - \\
&\qquad\qquad h[\mu] \sum_{r=1}^{\kappa} \omega_i^r[\mu] s_i^r[\mu], && \forall i, \mu && \text{(6c)} \\
&\qquad \sum_{r=1}^{\kappa} \omega_i^r[\mu] \leq 1, && \forall i, \mu && \text{(6d)}
\end{aligned}
$$

$$\sum_{i=1}^{n} \omega_i^r[\mu] \leq m_r, \qquad \forall r, \mu \tag{6e}$$

$$s_{min}^r \leq s_i^r[\mu] \leq s_{max}^r, \qquad \forall i, r, \mu \tag{6f}$$

$$0 \leq \omega_i^r[\mu] \leq 1, \qquad \forall i, r, \mu \tag{6g}$$

where $\omega_i^r[\mu]$ is defined as the percentage of the time interval $[\tau_\mu, \tau_{\mu+1}]$ for which task T_i is executing on a processor of type-r at speed $s_i^r[\mu]$. (6d) guarantees that a task will not run on more than one processor at a time. The constraint that the total workload at each time interval should be less than or equal to the system capacity is specified in (6e). Upper and lower bounds on task execution speed and percentage of task execution time are given in (6f) and (6g), respectively.

3.2.2 Solving the workload partitioning problem as a linear program (LP-DVFS)

The problem (6) can be further simplified to an LP if the set of speed levels S^r is finite, as is often the case for practical systems. We denote with s_q^r the execution speed at level $q \in Q^r := \{1, \ldots, l_r\}$ of an r-type processor, where l_r is the total number of speed levels of an r-type processor. Let $\forall q$ be short-hand for $\forall q \in Q^r$.

Consider now the following finite-dimensional LP:

find $x_i[\cdot], \omega_{iq}^r[\cdot], \ \forall i \in I, q \in Q^r, r \in R$

subject to

$$x_i[\Phi_b(T_i)] = \underline{x}_i, \qquad \forall i \tag{7a}$$

$$x_i[\mu] = 0, \qquad \forall i, \mu \notin \mathcal{U}_i \tag{7b}$$

$$x_i[\mu+1] \geq x_i[\mu] - h[\mu] \sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} \omega_{iq}^r[\mu] s_q^r, \qquad \forall i, \mu \tag{7c}$$

$$\sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} \omega_{iq}^r[\mu] \leq 1, \qquad \forall i, \mu \tag{7d}$$

$$\sum_{i=1}^{n} \sum_{q=1}^{l_r} \omega_{iq}^r[\mu] \leq m_r, \qquad \forall r, \mu \tag{7e}$$

$$0 \leq \omega_{iq}^r[\mu] \leq 1, \qquad \forall i, q, r, \mu \tag{7f}$$

where $\omega_{iq}^r[\mu]$ is the percentage of the time interval $[\tau_\mu, \tau_{\mu+1}]$ for which task T_i is executing on a processor of type-r at a speed level q. Note that all constraints are similar to (6), but the speed levels are fixed.

Theorem 3 *A solution to* (6) *can be constructed from a solution to* (7)*, and vice versa, if the discrete speed set* S^r *is any finite subset of the closed interval* $[s_{min}^r, s_{max}^r]$ *with* s_{min}^r *and* s_{max}^r *in* S^r *for all* r.

Proof Let $(\tilde{x}, \tilde{\omega}, \tilde{s})$ denote a solution to (6) and (x, ω) a solution to (7). The result follows by noting that one can choose $\lambda_q^r[\mu] \in [0, 1]$ such that $\sum_q \lambda_q^r[\mu] s_q^r[\mu] = \tilde{s}_i^r[\mu]$, $\omega_{iq}^r[\mu] = \lambda_q^r[\mu]\tilde{\omega}_i^r[\mu]$ and $\sum_q \lambda_q^r[\mu] = 1,\ \forall i, q, r, \mu$ are satisfied. □

3.2.3 Task ordering algorithm

This section discusses how to find a valid schedule in the task ordering step for each time interval $[\tau_\mu, \tau_{\mu+1}]$. Since the solutions obtained in the workload partitioning step are partitioning workloads of each task on each processor type within each time interval, one might think of using McNaughton's wrap around algorithm (McNaughton 1959) to find a valid schedule for each processor within the processor type. However, McNaughton's wrap around algorithm only guarantees that a task will not be executed at the same time within the cluster. There exists a possibility that a task will be assigned to more than one processor type (cluster) at the same time.

To avoid a parallel execution on any two clusters, we can adopt the Hetero-Wrap algorithm proposed in Chwa et al. (2015) to solve a task ordering problem of a two-type heterogeneous multiprocessor platform. The algorithm takes the workload partitioning solution to STEP 1 as its inputs and returns $(\sigma_{ik}^r, \eta_{ik}^r) \in [0, 1]^2,\ \forall i, k, r$, which is a task-to-processor interval assignment on each cluster. Note that, for a solution to problem (7), we define the total execution workload of a task $\omega_i^r := \sum_q \omega_{iq}^r$ and assume that the percentage of execution times of each task at all frequency levels ω_{iq}^r will be grouped together in order to minimize the number of migrations and preemptions. In order to be self-contained, the Hetero-Wrap algorithm is given in Algorithm 1. Specifically, the algorithm classifies the tasks into four subsets: (i) a set IM_a of migrating tasks with $\sum_r \omega_i^r = 1$, (ii) a set IM_b of migrating tasks with $\sum_r \omega_i^r < 1$, (iii) a set CP_1 of partitioned tasks on cluster of type-1, and (iv) a set CP_2 of partitioned tasks on cluster of type-2. The algorithm then employs the following simple rules:

- For a type-1 cluster, tasks are scheduled in the order of IM_a, IM_b and CP_1 using McNaughton's wrap around algorithm. That is, a slot along the number line is allocated, starting at zero, with the length equal to m_1 and the task is aligned with its assigned workload on empty slots of the cluster in the specified order starting from left to right.
- For a type-2 cluster, in the same manner, tasks are scheduled using McNaughton's wrap around algorithm, but in the order of IM_a, IM_b and CP_2 starting from right to left. Note that the order of tasks in IM_a has to be consistent with the order in a type-1 cluster.

However, the algorithm requires a feasible solution to (6) or (7), in which IM_b has at most one task, which we will call an inter-cluster migrating task. From Theorem 3, we can always transform a solution to (6) into a solution to (7). Therefore, we only need to show that there exists a solution to (7) with at most one inter-cluster migrating tasks that lies on the vertex of the feasible region by the following facts and lemma.

Fact 4 *Among all the solutions to an LP, at least one solution lies at a vertex of the feasible region. In other words, at least one solution is a basic solution.*

Algorithm 1 Hetero-Wrap Algorithm (Chwa et al. 2015)

INPUT: $\omega_i^r, m_r, \ \forall i, r$
$\sigma_{ik}^r \leftarrow 0, \ \eta_{ik}^r \leftarrow 1, \ \forall i, k, r$
$p_1 \leftarrow 0, \ p_2 \leftarrow m_2, \ k_1 \leftarrow 1, \ k_2 \leftarrow m_2$
for $r = 1, 2$ **do**
  if $r = 1$ **then**
    for $i \in \{IM_a, IM_b, CP_1\}$ **do**
      if $p_1 = 0$ **then**
        $\eta_{ik_1}^r \leftarrow w_i^r, \ p_1 \leftarrow \eta_{ik_1}^r$
      else
        if $p_1 + w_i^r \leq k_1$ **then**
          $\sigma_{ik_1}^r \leftarrow p_1 - (k_1 - 1)$
          $\eta_{ik_1}^r \leftarrow p_1 + w_i^r - (k_1 - 1)$
          $p_1 \leftarrow p_1 + w_i^r$
        else
          $\sigma_{ik_1}^r \leftarrow p_1 - (k - 1)$
          $\eta_{i(k_1+1)}^r \leftarrow p_1 + w_i^r - k_1$
          $k_1 \leftarrow k_1 + 1$
        end if
      end if
    end for
  else
    for $i \in \{IM_a, IM_b, CP_2\}$ **do**
      if $p_2 = m_2$ **then**
        $\sigma_{ik_2}^r \leftarrow 1 - \omega_i^r$
        $p_2 \leftarrow \sigma_{ik_2}^r$
      else
        if $p_2 - \omega_i^r \geq k_2 - 1$ **then**
          $\sigma_{ik_2}^r \leftarrow p_2 - \omega_i^r - (k_2 - 1)$
          $\eta_{ik_2}^r \leftarrow p_2 - (k_2 - 1)$
          $p_2 \leftarrow p_2 - \omega_i^r$
        else
          $\eta_{ik_2}^r \leftarrow p_2 - (k_2 - 1)$
          $k_2 \leftarrow k_2 - 1$
          $\sigma_{ik_2} \leftarrow p_2 - \omega_i^r - k_2 + 1$
        end if
      end if
    end for
  end if
end for
RETURN: $(\sigma_{ik}^r, \eta_{ik}^r) \in [0, 1]^2, \ \forall i, k, r$

Proof The Fundamental Theorem of Linear Programming, which states that if a feasible solution exists, then a basic feasible solution exists (Vanderbei 2001, p. 38). □

Fact 5 *A feasible solution to an LP that is not a basic solution can always be converted into a basic solution.*

Proof This follows from the Fundamental Theorem of Linear Programming (Vanderbei 2001, p. 38). □

Fact 6 (Baruah 2004, *Fact* 2) *Consider a linear program* $\min\{c^T\chi \mid A\chi \leq b, \chi \in \mathbb{R}^n\}$ *for some* $A \in \mathbb{R}^{(m+n)\times n}$, $b \in \mathbb{R}^{m+n}$, $c \in \mathbb{R}^n$. *Suppose that n constraints are nonnegative constraints on each variable, i.e.* $\chi_i \geq 0,\ \forall i \in \{1, 2, \ldots, n\}$ *and the rest are m linearly independent constraints. If* $m < n$, *then a basic solution will have at most m non-zero values.*

Proof A unique basic solution can be identified by any $n + m$ linearly independent active constraints. Since there are n nonnegative constraints and $m < n$, a basic solution will have at most m non-zero values. □

Lemma 1 *For a solution to* (7) *that lies on the vertex of the feasible region, there will be at most one inter-cluster partitioning task.*

Proof The number of variables ω subjected to nonnegative constraint (7f) at each time interval of (7) is $n(\sum_r l_r)$. The number of variables ω subjected to a set of necessary and sufficient feasibility constraints (7d)-(7e) is $n + 2$. Note that we do not count the number of variables in (7c) because (7c) and (7d) are linearly dependent constraints for a given value of $\xi_i[\mu] := (x_i[\mu] - x_i[\mu + 1])/h[\mu]$. If we assume that $n \geq 2$ and each processor type has at least one speed level, then it follows from Fact 6 that the number of non-zero values of variable ω, a solution to (7) at the vertex of the feasible region, is at most $n+2$. Let γ be the number of tasks assigned to two processor types. Therefore, there are $2\gamma + (n - \gamma)$ entries of variable ω that are non-zero. This implies that $\gamma < 2$, i.e. the number of inter-cluster partitioning tasks is at most one. □

To illustrate how Algorithm 1 works, consider a simple taskset in which the percentage of execution workload partition at time interval $[\tau_\mu, \tau_{\mu+1}]$ for each task is as shown in Table 1. A feasible schedule obtained by Algorithm 1 is shown in Fig. 2, where the numbers in a circle denote the order of tasks allocation. Notice that the processors of a cluster 1 are filled from left to right, while the processors of a cluster 2 are filled from right to left. For this example, $m_1 = m_2 = 2$, $IM_a = \{T_1, T_2\}$, $IM_b = \{T_3\}$, $CP_1 = \{T_4\}$ and $CP_2 = \{T_5\}$.

Theorem 7 *If a solution to* (1) *exists, then a solution to* (6)*/*(7) *exists. Furthermore, at least one valid schedule satisfying* (1) *can be constructed from a solution to problem* (6)*/*(7) *and the output from Algorithm 1.*

Proof The existence of a valid schedule is proven in Chwa et al. (2015, Thm 3). It follows from Facts 4–6 and Lemma 1 that one can compute a solution with at most one inter-cluster partitioning task. Given a solution to (6)/(7) and the output from Algorithm 1 for all intervals, choose a to be a step function such that $a^r_{ik}(t) = 1$ when $\sigma^r_{ik}[\mu, \nu]h[\mu, \nu] \leq t - \tau_{\mu,\nu} < \eta^r_{ik}[\mu, \nu]h[\mu, \nu + 1]$ and $a^r_{ik}(t) = 0$ otherwise, $\forall i, k, r, \mu, \nu$. Specifically, one can verify that the following condition holds

$$h[\mu, \nu]\omega^r_i[\mu] = \int_{\tau_{\mu,\nu}}^{\tau_{\mu,\nu+1}} \sum_k a^r_{ik}(t)dt,\ \forall i, r, \mu, \nu. \tag{8}$$

Then it is straightforward to show that (1) is satisfied. □

Table 1 Workload partition example

	T_1	T_2	T_3	T_4	T_5
ω_i^1	0.3	0.6	0.2	0.5	0
ω_i^2	0.7	0.4	0.4	0	0.5

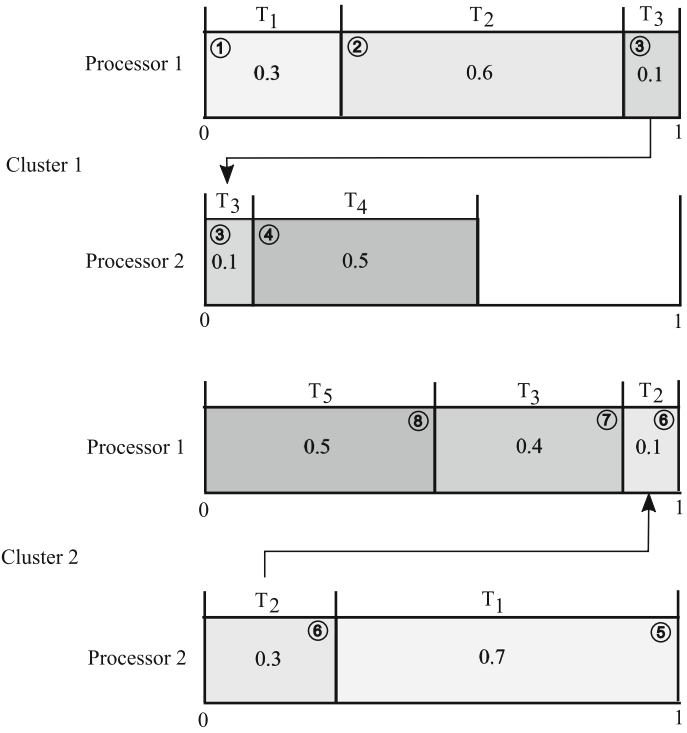


Fig. 2 A feasible task schedule according to Algorithm 1

3.3 Computational complexity

Note that, although we need to solve the same multiprocessor scheduling problem with two steps in this section, the computation times to solve (6) or (7) is extremely fast compared to solving problem (4), i.e. even for a small problem, the times to compute a solution of (4) can be up to an hour, while (6) or (7) can be solved in milliseconds using a general-purpose desktop PC with off-the-shelf optimization solvers. In particular, great advances in LP solvers have been made in the last three decades and an LP can be solved to any specified accuracy in polynomial time using a suitably-chosen interior point method (Boyd and Vandenberghe 2004). Moreover, the optimization problems defined in this section are highly structured with sparse matrices. These facts can be exploited to develop efficient tailor-made solvers, as in the literature on real-time optimization-based or predictive control (Betts 2010; Boyd and Vandenberghe 2004; Mayne 2014), which can outperform general-purpose software by

orders of magnitude. For example, custom optimization algorithms implemented on hardware accelerators, such as field-programmable gate arrays (FPGAs), can allow one to solve optimal control problems online in less than 1 μs (Jerez et al. 2014). The solution to the optimization problems can also be precomputed as a lookup table by using methods in parametric programming that have been developed in the control community (Bemporad et al. 2002a, b).

A more detailed complexity analysis is possible for the problems considered here. Since the time taken to solve the NLP (6) and the LP (7) is orders of magnitude less than solving the MINLP (4), we restrict ourselves to the NLP (6) and LP (7). For simplicity, assume that an interior point method is used to solve these problems.

Consider first the complexity of computing Newton (search) directions. Note that the number of decision variables and constraints grow $\mathcal{O}(nN)$, hence it follows that the time to compute a Newton direction increases $\mathcal{O}(n^3N^3)$. However, the problem here as a very particular structure and one can do better than this naive analysis. If care is taken with the formulation of the optimization problem and sparse linear solvers are employed as in Betts (2010) and Kang et al. (2014), then the number of non-zeros in the KKT matrix increases $\mathcal{O}(nN)$. Hence, a Lanczos-based iterative linear solver, such as MINRES, will be able to solve the sparse symmetric KKT system in $\mathcal{O}(n^2N^2)$ time. We note that this is only an upper bound and it is possible to exploit structure even further. Using a similar method as in Kang et al. (2014), in Kasture (2017)[3] it is shown that one can improve on this bound and that the Newton directions for the LP (7) can be computed in $\mathcal{O}(nN^2)$ time (note that $n \geq N$ for the problems in this paper). In practice, the method in Kasture (2017) scales better than this bound, suggesting that there is further structure to exploit; however, a detailed analysis is clearly beyond the scope of this paper and therefore left as an open research question.

Consider now the number of Newton steps required to compute a solution. It follows from established results that the number of Newton steps in an interior-point solver for an LP and certain classes of convex NLPs is bounded by $\mathcal{O}(\sqrt{nN})$ (Boyd and Vandenberghe 2004, Sect. 11.5). However, as is well-known, this bound is very conservative; in practice the number of steps hardly changes and is usually on the order of a few tens. This analysis clearly holds for the LP (7). However, we have so far been unable to derive a similar bound for the NLP (6), but have found that in practice the number of iterations of an interior-point solver is insensitive to the problem size if care is taken. An interesting open research question is therefore whether it is possible to reformulate the NLP (6) such that results on self-concordant functions (Boyd and Vandenberghe 2004, Sect. 11.5) can be used to derive a similar, scalable bound on the number of Newton iterations when solving the NLP (6).

Finally, note that the complexity of Algorithm 1 is $\mathcal{O}(n)$ (Chwa et al. 2015). Therefore, we expect that it would be feasible to execute the proposed algorithms online for certain taskets, but of course significant research on this topic still needs to be done in

[3] The work in Kasture 2017 was started and supervised by Eric Kerrigan after the original submission date of this paper. Please contact Kerrigan for a copy of thesis.

order to extend the scope and exploit the particular structure that arises in scheduling problems.

4 Energy optimality

4.1 Energy consumption model

A power consumption model can be expressed as a summation of dynamic power consumption P_d and static power consumption P_s. Dynamic power consumption is due to the charging and discharging of CMOS gates, while static power consumption is due to subthreshold leakage current and reverse bias junction current (Rabaey et al. 2003). The dynamic power consumption of CMOS processors at a clock frequency $f = sf_{max}$ is given by

$$P_d(s) = C_{ef} V_{dd}^2 s f_{max}, \tag{9a}$$

where the constraint

$$sf_{max} \leq \zeta \frac{(V_{dd} - V_t)^2}{V_{dd}} \tag{9b}$$

has to be satisfied (Rabaey et al. 2003). Here $C_{ef} > 0$ denotes the effective switch capacitance, V_{dd} is the supply voltage, V_t is the threshold voltage ($V_{dd} > V_t > 0\,\mathrm{V}$) and $\zeta > 0$ is a hardware-specific constant.

From (9b), it follows that if s increases, then the supply voltage V_{dd} may have to increase (and if V_{dd} decreases, so does s). In the literature, the total power consumption is often simply expressed as an increasing function of the form

$$P(s) := P_d(s) + P_s = \alpha s^\beta + P_s, \tag{10}$$

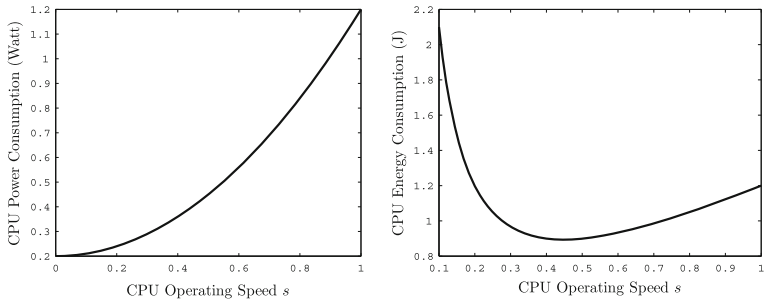
where $\alpha > 0$ and $\beta \geq 1$ are hardware-dependent constants, while the static power consumption P_s is assumed to be either constant or zero (Li and Wu 2013).

The energy consumption of executing and completing a task T_i at a constant speed s_i is given by

$$E(s_i) := \frac{c_i}{f_{max}} \frac{(P_d(s_i) + P_s)}{s_i} = \frac{x_i (P_d(s_i) + P_s)}{s_i}. \tag{11a}$$

In the literature, it is often assumed that E is an increasing function of the operating speed. However, because $s \mapsto 1/s$ is a decreasing function, it follows that the energy consumed might not be an increasing function if P_s is non-zero; Fig. 3 gives an example of when the energy is non-monotonic, even if the power is an increasing function of clock frequency.

This result implies the existence of a non-zero energy-efficient speed s_{eff}, i.e. the minimizer of (11) (Miyoshi et al. 2002; Chen et al. 2006; Aydin et al. 2006). Moreover, in the work of Zhai et al. (2004), the non-convex relationship between the



(a) $P(s) = s^2 + 0.2$ **(b)** $E(s) = (s^2 + 0.2)/s$

Fig. 3 Example of a non-increasing active energy function, but where the active power consumption is an increasing function

energy consumption and processor speed can be observed as a result of scaling supply voltage.

The total energy consumption of executing a real-time task T_i can be expressed as a summation of active energy consumption and idle energy consumption, i.e. $E = E_{active} + E_{idle}$, where E_{active} is the energy consumption when the processor is busy executing the task and E_{idle} is the energy consumption when the processor is idle. The energy consumption of executing and completing a task T_i at a constant speed s_i is

$$E(s_i) = E_{active}(s_i) + E_{idle} \tag{12a}$$

$$= \frac{c_i}{f_{max}} \frac{(P_{active}(s_i) - P_{idle})}{s_i} + P_{idle} d_i \tag{12b}$$

$$= \frac{\underline{x}_i (P_{active}(s_i) - P_{idle})}{s_i} + P_{idle} d_i, \tag{12c}$$

where $P_{active}(s) := P_{da}(s) + P_{sa}$ is the total power consumption in the active interval, $P_{idle} := P_{di} + P_{si}$ is the total power consumption during the idle period. $P_{da} > 0$ and $P_{sa} \geq 0$ are dynamic and static power consumption during the active period, respectively. Similarly, $P_{di} > 0$ and $P_{si} \geq 0$ are the dynamic and static power consumption during the idle period. P_{di} will be assumed to be a constant, since the processor is executing a nop (no operation) instruction at the lowest frequency f_{min} during the idle interval. P_{sa} and P_{si} are also assumed to be constants where $P_{si} < P_{sa}$. Note that $P_{active}(s) - P_{idle}$ is strictly greater than zero.

4.2 Optimality problem formulation

The scheduling problem with the objective to minimize the total energy consumption of executing the taskset on a two-type heterogeneous multiprocessor can be formulated as the following optimal control problems:

(I) Continuous Optimal Control Problem:

$$\underset{\substack{x_i(\cdot),a^r_{ik}(\cdot),s^r_k(\cdot),\\ \forall i\in I,k\in K^r,r\in R}}{\text{minimize}} \quad \sum_{r,k,i}\int_0^L \ell^r(a^r_{ik}(t),s^r_k(t))dt \tag{13}$$

subject to (1).

(II) MINLP-DVFS:

$$\underset{\substack{x_i[\cdot],a^r_{ik}[\cdot],s^r_k[\cdot],h[\cdot],\\ \forall i\in I,k\in K^r,r\in R}}{\text{minimize}} \quad \sum_{r,\mu,\nu,k,i} h[\mu,\nu]\ell^r(a^r_{ik}[\cdot],s^r_k[\cdot]) \tag{14}$$

subject to (4).

(III) NLP-DVFS:

$$\underset{\substack{x_i[\cdot],\omega^r_i[\cdot],s^r_i[\cdot],\\ \forall i\in I,r\in R}}{\text{minimize}} \quad \sum_{r,\mu,\nu,i} h[\mu]\ell^r(\omega^r_i[\cdot],s^r_i[\cdot]) \tag{15}$$

subject to 6.

(IV) LP-DVFS :

$$\underset{\substack{x_i[\cdot],\omega^r_{iq}[\cdot],\\ \forall i\in I,q\in Q^r,r\in R}}{\text{minimize}} \quad \sum_{r,\mu,\nu,i,q} h[\mu]\ell^r(\omega^r_{iq}[\cdot],s^r_q)$$

subject to (7). (16)

where $\ell^r(a,s) := a(P^r_{active}(s) - P^r_{idle})$. Note that (16) is an LP, since the cost is linear in the decision variables.

4.3 Constant or time-varying speed?

In this section, we present a result on a general speed selection trajectory for a uniprocessor scheduling problem with a real-time taskset. With this observation about optimal speed profile, we can formulate algorithms that are able to solve a more general class of scheduling problems than in the literature.

Consider the following simple example, illustrated in Fig. 4, where the power consumption model $P(\cdot)$ is a concave function of speed. Assume that s_2 is the lowest possible constant speed at which task T_i can be finished on time, i.e. $\underline{x}_i = s_2 d_i$. The energy consumed is $E(s_2) = P(s_2)d_i$ and the average power consumption $P(s_2) =: \bar{P}_c$. Let $\lambda \in [0,1]$ be a constant such that $s_2 = \lambda s_1 + (1-\lambda)s_3$, $s_1 < s_2 < s_3$. Suppose $s(\cdot)$ is a time-varying speed profile such that $s(t) = s_1$, $\forall t \in [0, t_1)$ and $s(t) = s_3$, $\forall t \in [t_1, d_i)$. We can choose t_1 such that $\underline{x}_i = s_1 t_1 + s_3(d_i - t_1)$. The energy used in this case is $E(s_1,s_3) = t_1 P(s_1) + (d_i - t_1)P(s_3)$. If we let $\lambda = t_1/d_i$, then the average power consumption $E(s_1,s_3)/d_i =: \bar{P}_{tv} = (t_1/d_i)P(s_1) + (1-(t_1/d_i))P(s_3) = \lambda P(s_1) + (1-\lambda)P(s_3)$. Since $P(\cdot)$ is concave, $P(s_2) \geq \lambda P(s_1) + (1-\lambda)P(s_3) = \bar{P}_{tv}$. This result implies that a time-varying speed profile is better than a constant speed profile when the power consumption is concave. Notably, the result can be generalised

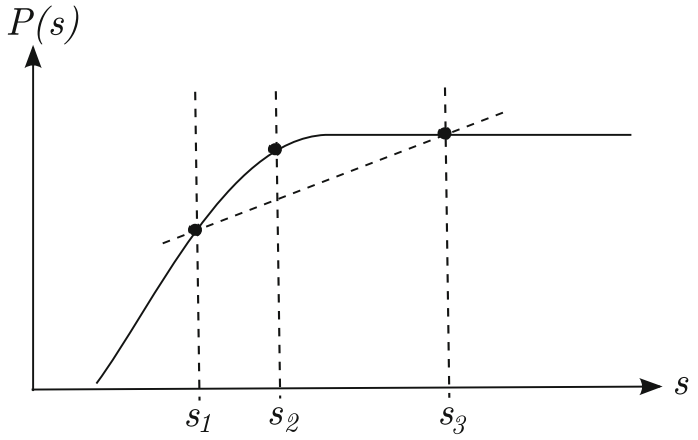


Fig. 4 A time-varying speed profile is better than a constant speed profile if the power function is not convex

to the case where the power model is non-convex, non-concave as well as discrete speed set.

Theorem 8 *Let a piecewise constant speed trajectory $s^*(\cdot)$ be given that maps every time instant in a closed interval $[t_0, t_f]$ to the domain S of a power function $P : S \to \mathbb{R}$. There exists a piecewise constant speed trajectory $s(\cdot)$ with at most one switch such that the amount of computations done and the energy consumed is the same as using $s^*(\cdot)$, i.e. $s(\cdot)$ is of the form*

$$s(t) := \begin{cases} \check{s} & \forall t \in [t_0, (t_f - t_0)\lambda + t_0) \\ \hat{s} & \forall t \in [t_0 + (t_f - t_0)\lambda, t_f] \end{cases}$$

where $\hat{s}, \check{s} \in S$, $\lambda \in [0, 1]$, such that the total amount of computations

$$c := \int_{t_0}^{t_f} s^*(t)dt = \int_{t_0}^{t_f} s(t)dt$$

and energy consumed

$$E := \int_{t_0}^{t_f} P(s^*(t))dt = \int_{t_0}^{t_f} P(s(t))dt.$$

Proof Let $\{\mathcal{T}_1, \ldots, \mathcal{T}_p\}$ be a partition of $[t_0, t_f]$ and range $s^* =: \{s_1, \ldots, s_p\} \subseteq S$ such that $s^*(t) = s_i$ for all $t \in \mathcal{T}_i$, $i \in \mathcal{I} := \{1, \ldots, p\}$. Define $\Delta_i := \int_{\mathcal{T}_i} dt$ as the size of the set $\mathcal{T}_i$ and $\lambda_i := \Delta_i/(t_f - t_0)$, $\forall i \in \mathcal{I}$.

It follows that $c = \sum_i s_i \Delta_i$ and $E = \sum_i P(s_i)\Delta_i$. Hence, the average speed $\bar{s} := c/(t_f - t_0) = \sum_i \lambda_i s_i$ and average power $\bar{P} := E/(t_f - t_0) = \sum_i \lambda_i P(s_i)$.

Note that $\sum_i \lambda_i = 1$. This implies that $(\bar{s}, \bar{P})$ is in the convex hull of the finite set $G := \{(s_i, P(s_i)) \in S \times \mathbb{R} \mid i \in \mathcal{I}\}$ with $\text{vert}(\text{conv}\, G) \subseteq G$. Hence, there

exists a $\lambda \in [0, 1]$ and two points $\hat{s}$ and $\check{s}$ in S with $(\hat{s}, P(\hat{s})) \in \text{vert}(\text{conv}\, G)$ and $(\check{s}, P(\check{s})) \in \text{vert}(\text{conv}\, G)$ such that $\bar{s} = \lambda\check{s} + (1-\lambda)\hat{s}$ and $\bar{P} = \lambda P(\check{s}) + (1-\lambda)P(\hat{s})$. If s is defined as above with these values of λ, $\hat{s}$ and $\check{s}$, then one can verify that $\int_{t_0}^{t_f} s(t)dt = (t_f - t_0)\bar{s}$ and $\int_{t_0}^{t_f} P(s(t))dt = (t_f - t_0)\bar{P}$. □

The following result has already been observed in Aydin et al. (2004, Prop. 1) and Gerards et al. (2014, Cor. 1).

Corollary 1 *Given a processor with a convex power model and required workload within a time interval, there exists a constant optimal speed profile if the set of speed levels S is a closed interval.*

Proof This is a special case of Theorem 8 and can be proven easily using Jensen's inequality. □

Corollary 2 *An optimal speed profile to* (13) *can be constructed by switching between no more than two non-zero speed levels within each time interval defined by two consecutive time steps of the major grid $\mathcal{T}$.*

Proof The overall optimal speed profile can be obtained by connecting an optimal time-varying speed profile proven in Theorem 8 for each partitioned time interval. Specifically, the generalised optimal speed profile is a step function. □

The result of the above Theorem and Corollaries can be applied directly to scheduling algorithms that adopt the DP technique such as, LLREF, DP-WRAP, as well as our algorithms in Sect. 3. Consider the problem of determining the optimal speeds at each time interval defined by two consecutive task deadlines. By subdividing time into such intervals, we can easily determine the optimal speed profile of four uniprocessor scheduling paradigms classified by power consumption and taskset models, i.e. (i) a convex power consumption model with implicit deadline taskset, (ii) a convex power consumption model with constrained deadline taskset, (iii) a non-convex power consumption model with implicit deadline taskset and (iv) a non-convex power consumption model with constrained deadline taskset. Specifically, if the taskset has an implicit deadline, then the required workloads (taskset density) are equal for all time intervals; the optimal speed profiles of all schedule intervals are the same as well. Therefore, the optimal speed profile is a constant for (i) (Cor. 1) and a combination of two speeds for (iii) (Cor. 8). However, for a constrained deadline taskset, the required workload varies from interval to interval, but is constant within the interval. Hence, even if the power function is (ii) convex or (iv) non-convex, the optimal speed profile is a (time-varying) piecewise constant function. In other words, for generality, a time-varying speed profile with two speed levels at each partitioned time interval is guaranteed to provide an energy optimal solution.

Theorem 9 *Consider the optimization problems* (13)–(16). *An optimal speed profile for* (13) *can be constructed using any of the following methods:*

– *Compute a solution to* (14) *with the lower bound on M at least twice the bound in Theorem 2.*

Table 2 ARM cortex-A15 (big) processor details (Chwa et al. 2014)

Voltage (V)	0.93	0.96	1.0	1.04	1.08	1.1	1.15	1.2	1.23
Freq. (MHz)	800	900	1000	1100	1200	1300	1400	1500	1600
Speed	0.5	0.5625	0.625	0.6875	0.75	0.8125	0.875	0.9375	1.0
Power (mW)	327	392	472	562	661	742	874	1019	1142

Table 3 ARM cortex-A7 (LITTLE) processor details (Chwa et al. 2014)

Voltage (V)	0.9	0.94	1.01	1.09	1.2
Freq. (MHz)	250	300	400	500	600
Speed	0.1563	0.1875	0.25	0.3125	0.375
Power (mW)	32	42	64	92	134

– *If the active power function P_{active} is convex and the speed level sets are closed intervals, compute a solution to* (15)*. If there is more than one inter-cluster partitioning task, then the (finite) range of the optimal speed profile should be used to define and compute a solution to* (16) *with at most one inter-cluster partitioning task. This process is concluded with Algorithm* 1.
– *If the speed level sets are finite, compute a solution to* (16) *with at most one inter-cluster partitioning task, followed with Algorithm* 1.

Proof Follows from the choices of selecting a and s as in the proofs of Theorem 2 and Theorem 7. The cost of all problems are then equal. □

5 Simulation results

5.1 System, processor and task models

The energy efficiency of solving the above optimization problems is evaluated on the ARM big.LITTLE architecture, where a big core provides faster execution times, but consumes more energy than a LITTLE core. The details of the ARM Cortex-A15 (big) and Cortex-A7 (LITTLE) core, which have been validated in Chwa et al. (2014), are given in Tables 2 and 3.

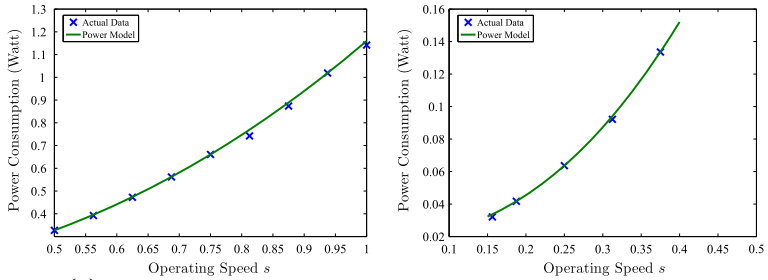
The active power consumption models, obtained by a polynomial curve fitting to the generic form (10), are shown in Table 4.

The plots of the actual data versus the fitted models are shown in Fig. 5. The idle power consumption was not reported, thus we will assume this to be a constant strictly less than the lowest active power consumption, namely $P_{idle} = 70$ mW for the big core and $P_{idle} = 12$ mW for the LITTLE core. To illustrate that our formulations are able to solve a broader class of multiprocessor scheduling problems than others optimal algorithms reported in the literature, we consider periodic taskset models with both implicit and constrained deadlines. However, a more general taskset model such as an arbitary deadline taskset, where the deadline could be greater than the period, a sporadic taskset model, where the inter-arrival time of successive tasks is at

Table 4 ARM processor power consumption models

Processor	Active power model	MAPE[a]
big	$P_{\text{active}}(s) := 1063.9s^{2.2} + 95.9075$	0.9283
LITTLE	$P_{\text{active}}(s) := 1103.17s^{2.3034} + 18.3549$	1.4131

[a] Mean Absolute Percentage Error (MAPE)
MAPE:= $\frac{1}{k}\sum_{i=1}^{k}\frac{|F(z(i))-y(i)|}{|y(i)|} \times 100$, where $\frac{|F(z(i))-y(i)|}{|y(i)|}$ is the magnitude of the relative error in the i^{th} measurement, $z \mapsto F(z)$ is the estimated function, z is the input data, y is the actual data and k is the total number of fitted points



(a) big core Processor **(b)** LITTLE core Processor

Fig. 5 Actual data versus fitted model

least p_i time units, and an aperiodic taskset can be solved by our algorithms as well. To guarantee the existence of a valid schedule, the minimum taskset density has to be less than or equal to the system capacity. Moreover, a periodic task needs to be able to be executed on any processor type. Specifically, the minimum task density should be less than or equal to the lowest capacity of all processor types, i.e. $\delta_i(1) \leq 0.375$ for this particular architecture.

5.2 Comparison between algorithms

For a system with a continuous speed range, four algorithms are compared: (i) MINLP-DVFS, (ii) NLP-DVFS, (iii) GWA-SVFS, which represents a global energy/feasibility-optimal workload allocation with constant frequency scaling scheme at a core-level and (iv) GWA-NoDVFS, which is a global scheduling approach without frequency scaling scheme. For a system with discrete speed levels, four algorithms are compared: (i) LP-DVFS, (ii) GWA-NoDVFS, (iii) GWA-DDiscrete and (iv) GWA-SDiscrete, which represent global energy/feasibility optimal workload allocation with time-varying and constant discrete frequency scaling schemes, respectively. Note that GWA-SVFS, GWA-NoDVFS, GWA-DDiscrete and GWA-SDiscrete are based on the mathematical optimization formulation proposed in Chwa et al. (2014), but adapted to our framework, for which details are given below.

GWA-SVFS/GWA-NoDVFS: Given m_r processors of type-r and n periodic tasks, determine a constant operating speed for each processor s_k^r and the workload ratio y_{ik}^r for all tasks within hyperperiod $\mathcal{L}$ that solves:

$$\underset{\substack{s_k^r, y_{ik}^r, \\ i\in I, k\in K^r, r\in R}}{\text{minimize}} \quad \sum_{r,i,k} \mathcal{L}_i \ell^r(\delta_{ik}^r(s_k^r), s_k^r) \tag{17a}$$

subject to

$$\sum_{r=1}^{\kappa}\sum_{k=1}^{m_r} y_{ik}^r = 1, \qquad \forall i \tag{17b}$$

$$\sum_{r=1}^{\kappa}\sum_{k=1}^{m_r} \delta_{ik}^r(s_k^r) \le 1, \qquad \forall i \tag{17c}$$

$$\sum_{i=1}^{n} \delta_{ik}^r(s_k^r) \le 1, \qquad \forall k, r \tag{17d}$$

$$0 \le y_{ik}^r \le 1, \qquad \forall i, k, r \tag{17e}$$

$$s_{min}^r \le s_k^r \le s_{max}^r, \qquad \forall k, r \qquad \text{(GWA-SVFS)} \tag{17f}$$

$$s_k^r = s_{max}^r, \qquad \forall k, r \qquad \text{(GWA-NoDVFS)} \tag{17g}$$

where y_{ik}^r is the ratio of the workload of task T_i on processor k of type-r, $\delta_{ik}^r(s_k^r)$ is the task density on processor k type-r defined as $\delta_{ik}^r(s_k^r) := y_{ik}^r c_i/(s_k^r f_{max} \min\{d_i, p_i\})$ and $\mathcal{L}_i := \mathcal{L}\min\{d_i, p_i\}/p_i$. Note that when $d_i = p_i$ as in the case of an implicit deadline taskset $\mathcal{L}_i = \mathcal{L}$, $\forall i$. (17b) ensures that all tasks will be allocated the amount of required execution time. The constraint that a task will not be executed on more than one processor at the same time is specified in (17c). (17d) asserts that the assigned workload will not exceed processor type capacity. Upper and lower bounds on the workload ratio of a task are given in (17e). The difference between GWA-SVFS and GWA-NoDVFS lies in restricting a core-level operating speed s_k^r to be either a continuous variable (17f) or fixed at the maximum value (17g).

GWA-DDiscrete: Given m_r processors of type-r and n periodic tasks, determine a percentage of the task workload y_{iq}^r at a specific speed level for all tasks within hyperperiod $\mathcal{L}$ that solves:

$$\underset{\substack{y_{iq}^r \\ i\in I, q\in Q^r, r\in R}}{\text{minimize}} \quad \sum_{r,i,q} \mathcal{L}_i \ell^r(\delta_{iq}^r(y_{iq}^r), s_q^r) \tag{18a}$$

subject to

$$\sum_{r=1}^{\kappa}\sum_{q=1}^{l_r} y_{iq}^r = 1, \qquad \forall i \tag{18b}$$

$$\sum_{r=1}^{\kappa}\sum_{q=1}^{l_r} \delta_{iq}^r(y_{iq}^r) \le 1, \qquad \forall i \tag{18c}$$

$$\sum_{i=1}^{n}\sum_{q=1}^{l_r} \delta_{iq}^r(y_{iq}^r) \le m_r, \qquad \forall r \tag{18d}$$

$$0 \le y_{iq}^r \le 1, \qquad \forall i, q, r \tag{18e}$$

where y^r_{iq} is the percentage of workload of task T_i on processor type-r at speed level q, $\delta^r_{iq}(y^r_{iq})$ is the task density on processor type-r at speed level q, i.e. $\delta^r_{iq}(y^r_{iq}) := y^r_{iq} c_i/(s^r_q f_{max} \min\{d_i, p_i\})$. Constraint (18b) guarantees that the total execution workload of a task is allocated. Constraint (18c) assures that a task will be executed only on one processor at a time. Constraint (18d) ensures that each processor type workload capacity is not violated. Constraint (18e) provides upper and lower bounds on a percentage of task workload at specific speed level.

GWA-SDiscrete: Given m_r processors of type-r and n periodic tasks, determine a percentage of task workload y^r_{iq} at a specific speed level and a processor speed level selection z^r_q for all tasks within hyperperiod $\mathcal{L}$ that solves:

$$\underset{\substack{y^r_{ikq}, z^r_{kq} \\ i\in I, k\in K^r, q\in Q^r, r\in R}}{\text{minimize}} \quad \sum_{r,i,q} \mathcal{L}_i \ell^r(\delta^r_{ikq}(y^r_{ikq}) z^r_{kq}, s^r_q) \tag{19a}$$

subject to

$$\sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} \sum_{k=1}^{m_r} y^r_{ikq} z^r_{kq} = 1, \qquad \forall i \tag{19b}$$

$$\sum_{q=1}^{l_r} z^r_{kq} = 1, \qquad \forall k, r \tag{19c}$$

$$\sum_{r=1}^{\kappa} \sum_{q=1}^{l_r} \delta^r_{ikq}(y^r_{ikq}) z^r_{kq} \le 1, \qquad \forall i \tag{19d}$$

$$\sum_{i=1}^{n} \sum_{q=1}^{l_r} \delta^r_{ikq}(y^r_{ikq}) z^r_{kq} \le 1, \qquad \forall k, r \tag{19e}$$

$$0 \le y^r_{ikq} \le 1, \qquad \forall i, k, q, r \tag{19f}$$

$$z^r_{kq} \in \{0, 1\}, \qquad \forall k, q, r \tag{19g}$$

where y^r_{ikq} is the workload partition of task T_i of processor k of an r-type at speed level q, z^r_{kq} is a speed level selection variable for processor k of an r-type , i.e. $z^r_{kq} = 1$ if a speed level q of an r-type processor is selected and $z^r_{kq} = 0$ otherwise. Constraint (19b), (19d)–(19f) are the same as the GWA-DDiscrete. Constraint (19c) assures that only one speed level is selected. Constraint (19g) emphasises that the speed level selection variable is a binary.

Note that GWA-SVFS and GWA-NoDVFS are NLPs, GWA-DDiscrete is an LP and GWA-SDiscrete is an MINLP. Moreover, the formulation of GWA-DDiscrete allows a processor to run with a time-varying combination of constant discrete speed levels, while GWA-SVFS and GWA-SDiscrete only allow a constant execution speed for each processor.

5.3 Simulation setup and results

For simplicity and without loss of generality, consider the case where independent real-time tasks are to be executed on two-type processor architectures, for which the details are given in Sect. 5.1. The MINLP formulations were modelled using ZIMPL (Koch 2004) and solved with SCIP (Achterberg 2009). The LP and NLP formulations were solved with SoPlex (Wunderling 1996) and Ipopt (Wächter and Biegler 2006), respectively. The value of the minor grid discretization step M is chosen according to Theorem 2.

For implict deadline tasksets, we consider the system composed of two big cores and six LITTLE cores, which has a system capacity of 4.25=(2+2.25). The total energy consumption of each taskset with a minimum taskset density varying from 0.5 to system capacity with a step of 0.25, given in Table 5, are evaluated. Note that, the first parameter of a task is $\underline{x}_i$; c_i can be obtained by multiplying $\underline{x}_i$ by f_{max}. Since the task period is the same as the deadline, the last parameter of the task model is dropped.

Figure 6a shows simulation results for scheduling a real-time taskset with implicit deadlines on an ideal system. The minimum taskset density D is represented on the horizontal axis. The vertical axis is the total energy consumption normalised by GWA-NoDVFS, where less than 1 means the algorithm does better than GWA-NoDVFS.

The three algorithms with a DVFS scheme, i.e. MINLP-DVFS, NLP-DVFS, and GWA-SVFS, produce the same optimal energy consumption, though both of our algorithms allow the operating speed to vary with time compared with a constant frequency scaling scheme, used by GWA-SVFS. The simulation results suggest that the optimal speed is a constant, rather than time-varying, for an implicit deadline taskset that has a constant workload over time. This result complies with Corollary 1. Moreover, the little core, which only has 37.5% computing power compared with the big core and consumes considerably less power even when running at full speed, will be selected by the optimizer before considering the big cores. This is why we can see two upwards parabolic curves in the figures, where the first one corresponds to the case where only little cores in the system are selected, while both core types are selected in the second, which happens when the minimum taskset density is larger than the little-core cluster's capacity.

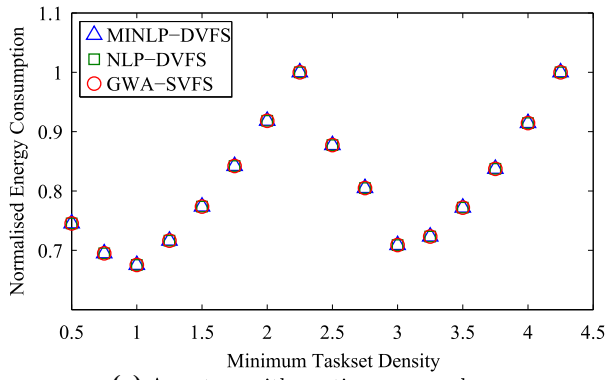
However, for a practical system, where a processor has discrete speed levels, the constant speed assignment is not an optimal strategy. As can be observed in Fig. 6b, the LP-DVFS and GWA-DDiscrete are energy optimal, while the GWA-SDiscrete is not. The results imply that to obtain an energy optimal schedule, a time-varying combination of discrete speed levels is necessary.

For a real-time taskset with constrained deadlines, we consider a system with one big core and one LITTLE core, i.e. a system capacity of $1.375 = (1 + 0.375)$. The simulation results of executing each taskset, listed in Table 6, are shown in Fig. 7, where the total energy consumption normalised by GWA-NoDVFS is on the vertical axis.

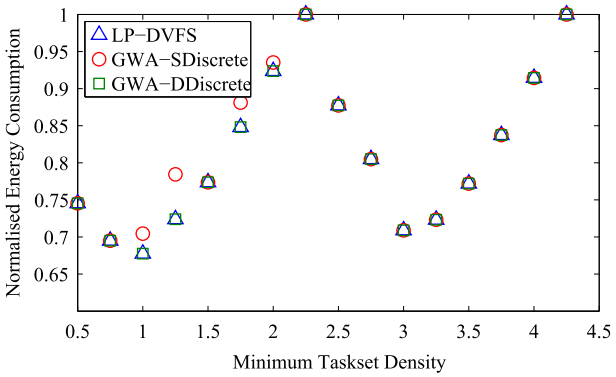
It can be seen from the plots that for a taskset with a piecewise constant and time-varying workload, i.e. constrained deadlines, our algorithms consume less energy than that of GWA-SVFS, GWA-DDiscrete and GWA-SDiscrete. This is because time

Table 5 Implicit deadline tasksets for simulation

D	Taskset
0.50	(1,5), (1,10), (4,20)
0.75	(1,5), (1,10), (5,20), (4,20)
1.00	(1,5), (1,10), (7,20), (7,20)
1.25	(1,5), (1,10), (6,20), (6,20), (7,20)
1.50	(1,5), (3,10), (6,20), (7,20), (7,20)
1.75	(1,5), (2,10), (6,20), (7,20), (7,20), (7,20)
2.00	(1,5), (3,10), (7,20), (7,20), (7,20), (7,20), (2,20)
2.25	(1,5), (3.5,10), (3.5,10), (6,20), (7,20), (7,20), (7,20)
2.50	(1,5), (3,10), (3,10), (3,10), (7,20), (7,20), (7,20), (7,20)
2.75	(1,5), (3.5,10), (3.5,10), (3,10), (7,20), (7,20), (7,20), (7,20), (3,20)
3.00	(1,5), (3,10), (3,10), (3,10), (3,10), (7,20), (7,20), (7,20), (7,20), (4,20)
3.25	(1,5), (3.5,10), (3.5,10), (3.5,10), (3.5,10), (7,20), (7,20), (7,20), (7,20), (5,20)
3.50	(1,5), (3.5,10), (3.5,10), (3,10), (3,10), (7,20), (7,20), (7,20), (7,20), (7,20), (5,20)
3.75	(1,5), (3.5,10), (3.5,10), (3,10), (3,10), (7.5,20), (7.5,20), (7.5,20), (7.5,20), (7.5,20), (7.5,20)
4.00	(1,5), (3.5,10), (3.5,10), (3,10), (3,10), (7.5,20), (7.5,20), (7.5,20), (7.5,20), (7.5,20), (7.5,20), (5,20)
4.25	(1,5), (3.75,10), (3.75,10), (3.75,10), (3.75,10), (7.5,20), (7.5,20), (7.5,20), (7.5,20), (7.5,20), (7.5,20), (6,20)



(a) A system with continuous speed range

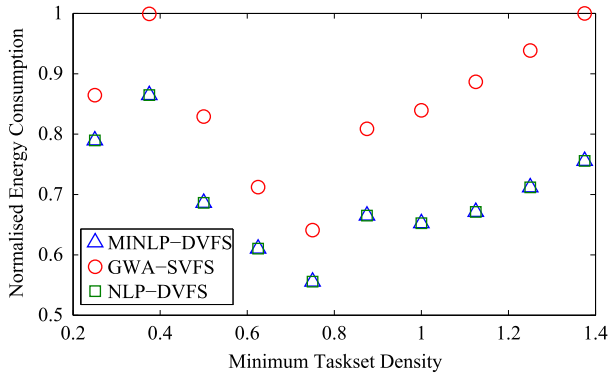


(b) A system with discrete speed levels

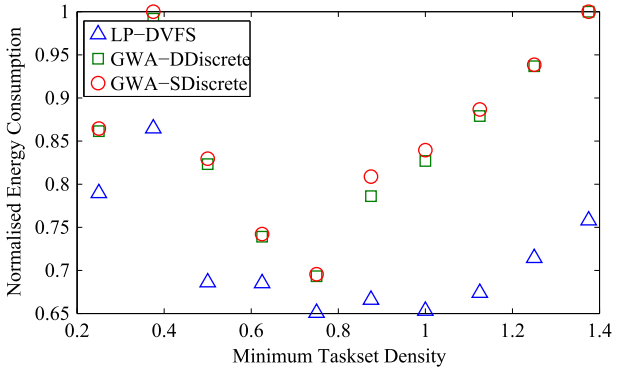
Fig. 6 Simulation results for scheduling real-time tasks with implicit deadlines

Table 6 Constrained deadline tasksets for simulation

D	Taskset
0.250	(0.9375,5,10), (0.625,10,10)
0.375	(1.5625,5,10), (0.625,10,10)
0.500	(1.875,5,10), (1.25,10,10)
0.625	(1.875,5,10), (1,5,10), (0.5,10,10)
0.750	(1.875,5,10), (1.625,5,10), (0.5,10,10)
0.875	(1.875,5,40), (1.75,5,40), (6,40,40)
1.000	(1.875,5,40), (1.875,5,40), (0.5,5,40), (6,40,40)
1.125	(1.875,5,40), (1.5,5,40), (1.3125,5,40), (6,40,40), (1.5,40,40)
1.25	(1.875,5,40), (1.875,5,40), (1.5625,5,40), (6,40,40), (1.5,40,40)
1.375	(1.875,5,40), (1.875,5,40), (1.875,5,40), (6,40,40), (4,40,40)



(a) A system with continuous speed range



(b) A system with discrete speed levels

Fig. 7 Simulation results for scheduling real-time tasks with constrained deadlines

is incorporated in our formulations, which provides benefits for solving a scheduling problem with a time-varying workload as well as a constant workload.

It has to be mentioned that the energy saving percentage varies with the taskset, which implies that the number on the plots shown here can be varied, but the significant outcomes stay the same. An example of how large the variability is illustrated in Fig. 8, where 5 constrained deadlines are randomly generated for each taskset density and are scheduled on a practical system composed of one of each processor type.

Lastly, although we had shown simulations with small tasksets in order to facilitate replicability, our proposed scheduling algorithms are scalable to large problems. (6)/(15) and (7)/(16) with large number of decision variables can be solved in milliseconds using a general-purpose PC with the off-the-self optimization solvers (Hei et al. 2008).

6 Conclusions

This work presents multiprocessor scheduling as an optimal control problem with the objective of minimizing the total energy consumption. We have shown that the scheduling problem is computationally tractable by first solving a workload parti-

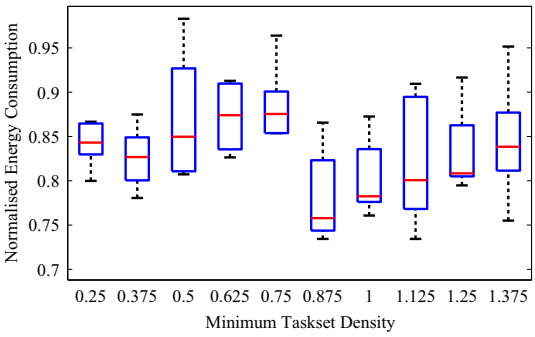


Fig. 8 Energy saving percentage box-plots of executing constrained deadline tasksets on a practical system

tioning problem, then a task ordering problem. The simulation results illustrate that our algorithms are both feasibility optimal and energy optimal when compared to an existing global energy/feasibility optimal workload allocation algorithm. Moreover, we have shown via proof and simulation that a constant frequency scaling scheme is enough to guarantee optimal energy consumption for an ideal system with a constant workload and convex power function, while this is not true in the case of a time-varying workload or a non-convex power function. For a practical system with discrete speed levels, a time-varying speed assignment is necessary to obtain an optimal energy consumption in general.

For future work, one could incorporate a DPM scheme and formulate the problem as a multi-objective optimization problem to further reduce energy consumption of a system. Extending the idea presented here to cope with uncertainty in a task's execution time using feedback is also possible, as in our work on homogeneous multiprocessor systems (Thammawichai and Kerrigan 2016). Though our work has been focused on minimizing the energy consumption, the framework could be easily applied to other objectives such as leakage-aware, thermal-aware and communication-aware scheduling problems. Numerically efficient methods could also be developed to solve optimization problems defined here, since they are highly structured with sparse matrices. Lastly, although our proposed multiprocessor scheduling framework is designed for a real-time system, it can also be applied at a higher level of abstraction, such as a data center, server farm or cloud computing, as well as other generic scheduling problems such as multi-stage multi-machine problems.

References

Achterberg T (2009) SCIP: solving constraint integer programs. Math Program Comput 1(1):1–41

ARM (2013) big.little technology: the future of mobile. http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

Awan M, Petters S (2013) Energy-aware partitioning of tasks onto a heterogeneous multi-core platform. In: 2013 IEEE 19th Real-time and embedded technology and applications symposium (RTAS), pp 205–214. doi:10.1109/RTAS.2013.6531093

Aydin H, Melhem R, Mosse D, Mejia-Alvarez P (2004) Power-aware scheduling for periodic real-time tasks. IEEE Trans Comput 53(5):584–600. doi:10.1109/TC.2004.1275298

Aydin H, Devadas V, Zhu D (2006) System-level energy management for periodic real-time tasks. In: 27th IEEE international real-time systems symposium, 2006. RTSS '06, pp 313–322. doi:10.1109/RTSS.2006.48

Baruah S (2004) Task partitioning upon heterogeneous multiprocessor platforms. In: 10th IEEE real-time and embedded technology and applications symposium, 2004. Proceedings. RTAS 2004, pp 536–543. doi:10.1109/RTTAS.2004.1317301

Baruah SK, Cohen NK, Plaxton CG, Varvel DA (1993) Proportionate progress: a notion of fairness in resource allocation. In: Proceedings of the twenty-fifth annual ACM symposium on theory of computing, STOC '93. ACM, New York, pp 345–354. doi:10.1145/167088.167194

Bemporad A, Borrelli F, Morari M (2002a) Model predictive control based on linear programming—the explicit solution. IEEE Trans Autom Control 47(12):1974–1985

Bemporad A, Morari M, Dua V, Pistikopoulos EN (2002b) The explicit linear quadratic regulator for constrained systems. Automatica 38(1):3–20

Betts J (2010) Practical methods for optimal control and estimation using nonlinear programming, 2nd edn. Society for Industrial and Applied Mathematics. doi:10.1137/1.9780898718577, http://epubs.siam.org/doi/abs/10.1137/1.9780898718577

Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press, New York

Chen JJ, Kuo CF (2007) Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In: 13th IEEE international conference on embedded and real-time computing systems and applications, 2007. RTCSA 2007, pp 28–38. doi:10.1109/RTCSA.2007.37

Chen JJ, Hsu HR, Kuo TW (2006) Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In: Proceedings of the 12th IEEE real-time and embedded technology and applications symposium, 2006, pp 408–417. doi:10.1109/RTAS.2006.25

Chen JJ, Yang CY, Lu HI, Kuo TW (2008) Approximation algorithms for multiprocessor energy-efficient scheduling of periodic real-time tasks with uncertain task execution time. In: Real-time and embedded technology and applications symposium, 2008. RTAS '08. IEEE, pp 13–23. doi:10.1109/RTAS.2008.24

Chen JJ, Schranzhofer A, Thiele L (2009) Energy minimization for periodic real-time tasks on heterogeneous processing units. In: IEEE international symposium on parallel distributed processing, 2009. IPDPS 2009, pp 1–12. doi:10.1109/IPDPS.2009.5161024

Cho H, Ravindran B, Jensen E (2006) An optimal real-time scheduling algorithm for multiprocessors. In: 27th IEEE international real-time systems symposium, 2006. RTSS '06, pp 101–110. doi:10.1109/RTSS.2006.10

Chwa HS, Seo J, Yoo H, Lee J, Shin I (2014) Energy and feasibility optimal global scheduling framework on big.little platforms. Tech. rep., Department of Computer Science, KAIST and Department of Computer Science and Engineering, Sungkyunkwan University, Republic of Korea. https://cs.kaist.ac.kr/upload_files/report/1407392146.pdf

Chwa HS, Seo J, Lee J, Shin I (2015) Optimal real-time scheduling on two-type heterogeneous multicore platforms. In: 2015 IEEE real-time systems symposium, pp 119–129

Funaoka K, Kato S, Yamasaki N (2008a) Energy-efficient optimal real-time scheduling on multiprocessors. In: 11th IEEE international symposium on object oriented real-time distributed computing (ISORC), 2008, pp 23–30. doi:10.1109/ISORC.2008.19

Funaoka K, Takeda A, Kato S, Yamasaki N (2008b) Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors. In: International symposium on industrial embedded systems, 2008. SIES 2008, pp 27–33. doi:10.1109/SIES.2008.4577677

Funk S, Berten V, Ho C, Goossens J (2012) A global optimal scheduling algorithm for multiprocessor low-power platforms. In: Proceedings of the 20th international conference on real-time and network systems. RTNS '12. ACM, New York, pp 71–80. doi:10.1145/2392987.2392996

Gerards M, Hurink J, Holzenspies P, Kuper J, Smit G (2014) Analytic clock frequency selection for global dvfs. In: 2014 22nd Euromicro international conference on parallel, distributed and network-based processing (PDP), pp 512–519. doi:10.1109/PDP.2014.103

Gerdts M (2006) A variable time transformation method for mixed-integer optimal control problems. Optim Control Appl Methods 27(3):169–182. doi:10.1002/oca.778

Goh LK, Veeravalli B, Viswanathan S (2009) Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems. IEEE Trans Parallel Distrib Syst 20(1):1–12. doi:10.1109/TPDS.2008.55

Hei L, Nocedal J, Waltz RA (2008) A numerical study of active-set and interior-point methods for bound constrained optimization. In: Bock HG, Kostina E, Phu HX, Rannacher R (eds) Modeling, simulation and optimization of complex processes: proceedings of the third international conference on high performance scientific computing, March 6–10, 2006. Springer, Berlin, pp 273–292. doi:10.1007/978-3-540-79409-7_18

Jerez JL, Goulart PJ, Richter S, Constantinides GA, Kerrigan EC, Morari M (2014) Embedded online optimization for model predictive control at megahertz rates. IEEE Trans Autom Control 59(12):3238–3251

Kang J, Cao Y, Word DP, Laird CD (2014) An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition. Comput Chem Eng 71:563–573

Kasture A (2017) Optimal control and scheduling of computing systems. Master's thesis, Imperial College London

Koch T (2004) Rapid mathematical prototyping. PhD thesis, Technische Universität, Berlin

Lawler E (1983) Recent results in the theory of machine scheduling. In: Bachem A, Korte B, Grötschel M (eds) Mathematical programming the state of the art. Springer, Berlin, pp 202–234. doi:10.1007/978-3-642-68874-4_9

Leung LF, Tsui CY, Ki WH (2004) Minimizing energy consumption of multiple-processors-core systems with simultaneous task allocation, scheduling and voltage assignment. In: Design automation conference, 2004. Proceedings of the ASP-DAC 2004. Asia and South Pacific, pp 647–652. doi:10.1109/ASPDAC.2004.1337672

Levin G, Funk S, Sadowski C, Pye I, Brandt S (2010) DP-FAIR: a simple model for understanding optimal multiprocessor scheduling. In: 22nd Euromicro conference on real-time systems (ECRTS), pp 3–13. doi:10.1109/ECRTS.2010.34

Li D, Wu J (2012) Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms. In: 41st International conference on parallel processing (ICPP), pp 430–439. doi:10.1109/ICPP.2012.26

Li D, Wu J (2013) Energy-aware scheduling on multiprocessor platforms. Springer briefs in computer science. Springer, New York

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61. doi:10.1145/321738.321743

Mayne DQ (2014) Model predictive control: recent developments and future promise. Automatica 50(12):2967–2986. doi:10.1016/j.automatica.2014.10.128

McNaughton R (1959) Scheduling with deadlines and loss function. Mach Sci 6(1):1–12

Miyoshi A, Lefurgy C, Van Hensbergen E, Rajamony R, Rajkumar R (2002) Critical power slope: Understanding the runtime effects of frequency scaling. In: Proceedings of the 16th international conference on supercomputing, ICS '02. ACM, New York, pp 35–44. doi:10.1145/514191.514200

Nelissen G, Berten V, Nelis V, Goossens J, Milojevic D (2012) U-edf: an unfair but optimal multiprocessor scheduling algorithm for sporadic tasks. In: 24th Euromicro conference on real-time systems (ECRTS), pp 13–23. doi:10.1109/ECRTS.2012.36

Rabaey JM, Chandrakasan AP, Nikolic B (2003) Digital integrated circuits: a design perspective, 2nd edn. Prentice Hall electronics and VLSI series. Pearson Education, Upper Saddle River, NJ

Regnier P, Lima G, Massa E, Levin G, Brandt S (2011) Run: optimal multiprocessor real-time scheduling via reduction to uniprocessor. In: IEEE 32nd real-time systems symposium (RTSS), pp 104–115. doi:10.1109/RTSS.2011.17

Thammawichai M, Kerrigan E (2016) Feedback scheduling for energy-efficient real-time homogeneous multiprocessor systems. In: Proc. 55th IEEE conference on decision and control, Las Vegas, USA

Ullman J (1975) NP-complete scheduling problems. J Comput Syst Sci 10(3):384–393. doi:10.1016/S0022-0000(75)80008-0

Vanderbei RJ (2001) Linear programming: foundations and extensions. International series in operations research & management science. Kluwer Academic, Boston. http://opac.inria.fr/record=b1100407

Wächter A, Biegler LT (2006) On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math Program 106(1):25–57. doi:10.1007/s10107-004-0559-y

Wu F, Jin S, Wang Y (2012) A simple model for the energy-efficient optimal real-time multiprocessor scheduling. In: IEEE international conference on computer science and automation engineering (CSAE), vol 3, pp 18–21. doi:10.1109/CSAE.2012.6272898

Wunderling R (1996) Paralleler und objektorientierter simplex-algorithmus. PhD thesis, Technische Universität, Berlin

Xian C, Lu YH, Li Z (2007) Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In: 44th ACM/IEEE design automation conference, 2007. DAC '07, pp 664–669

Yang CY, Chen JJ, Kuo TW, Thiele L (2009) An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. In: Design, automation test in Europe conference exhibition, 2009. DATE '09, pp 694–699. doi:10.1109/DATE.2009.5090754

Yu Y, Prasanna V (2002) Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In: Ninth international conference on parallel and distributed systems, pp 341–348. doi:10.1109/ICPADS.2002.1183422

Zhai B, Blaauw D, Sylvester D, Flautner K (2004) Theoretical and practical limits of dynamic voltage scaling. In: Proceedings of the 41st annual design automation conference, DAC '04. ACM, New York, pp 868–873. doi:10.1145/996566.996798

Mason Thammawichai received the BS degree in Computer Engineering from University of Wisconsin-Madison, USA and the MSc in Avionic Systems from University of Sheffield, UK. He finished his PhD from Imperial College London in 2016. His main areas of research are real-time scheduling, mathematical optimization and optimal control.

Eric C. Kerrigan (S'94-M'02) received a PhD from the University of Cambridge in 2001 and has been a faculty member at Imperial College London since 2006. His research is on efficient numerical methods and computing architectures for solving advanced optimization, control and estimation problems in aerospace, renewable energy and computing systems. He is on the IEEE Control Systems Society Conference Editorial Board and is an associate editor of the IEEE Transactions on Control Systems Technology and Control Engineering Practice.