Energy-aware Scheduling of Multi-version Tasks on Heterogeneous Real-time Systems

Julius Roeder
University of Amsterdam
Amsterdam, Netherlands
j.roeder@uva.nl

Benjamin Rouxel
University of Amsterdam
Amsterdam, Netherlands
b.rouxel@uva.nl

Sebastian Altmeyer
University of Augsburg
Augsburg, Germany
altmeyer@informatik.uni-augsburg.de

Clemens Grelck
University of Amsterdam
Amsterdam, Netherlands
c.grelck@uva.nl

ABSTRACT

The emergence of battery-powered devices has led to an increase of interest in the energy consumption of computing devices. For embedded systems, dispatching the workload on different computing units enables the optimisation of the overall energy consumption on high-performance heterogeneous platforms. However, to use the full power of heterogeneity, architecture specific binary blocks are required, each with different energy/time trade-offs. Finding a scheduling strategy that minimises the energy consumption, while guaranteeing timing constraints creates new challenges. These challenges can only be met by using the full heterogeneous capacity of the platform (e.g. heterogeneous CPU, GPU, DVFS, dynamic frequency changes from within an application).

We propose an off-line scheduling algorithm for dependent multi-version tasks based on Forward List Scheduling to minimise the overall energy consumption. Our heuristic accounts for Dynamic Voltage and Frequency Scaling (DVFS) and enables applications to dynamically adapt voltage and frequency during run time. We demonstrate the benefits of multi-version task models coupled with an energy-aware scheduler. We observe that selecting the most energy efficient version for each task does not lead to the lowest energy consumption for the whole application. Then we show that our approach produces schedules that are on average 45.6% more energy efficient than schedules produced by a state-of-the-art scheduling algorithm. Next we compare our heuristic against an optimal solution derived by an Integer Linear Programming (ILP) formulation (deviation of 1.6% on average). Lastly, we empirically show that the energy consumption predicted by our scheduler is close to the actual measured energy consumption on a Odroid-XU4 board (at most -15.8%).

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8104-8
https://doi.org/10.1145/3412841.3441930

CCS CONCEPTS

• **Computer systems organization** → **Embedded software**; *Multicore architectures*; • **Hardware** → *Platform power issues*;

KEYWORDS

DAG, energy-aware scheduling, multi-version, DVFS

ACM Reference Format:
Julius Roeder, Benjamin Rouxel, Sebastian Altmeyer, and Clemens Grelck. 2021. Energy-aware Scheduling of Multi-version Tasks on Heterogeneous Real-time Systems. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3412841.3441930

1 INTRODUCTION

Systems such as autonomous robots and unmanned aerial vehicles (UAV) used to be limited in terms of computation power, as the corresponding software stack required powerful workstations (e.g. for image analysis). This limitation is now relaxed with the availability of high-performance embedded computers, such as Odroid-XU4 [2] or Nvidia Jetson boards [1]. Nonetheless, this type of systems are often battery-powered, e.g [6, 16, 22], making energy consumption an important design criterion [32]. The challenge is to minimise the overall energy consumption while guaranteeing timing constraints on complex, heterogeneous architectures.

To tackle this challenge, we need to fully utilise the heterogeneous capacity of the hardware. This among others includes heterogeneous CPUs (e.g. big.LITTLE) and accelerators (e.g. GPUs). Different compute units may require architecture-dependent binaries (e.g. CPU vs. GPU) due to different instruction set architectures (ISA). The absence of binary compatibility makes multi-version tasks a natural, if not necessary, starting point for our work on scheduling for modern heterogeneous embedded platforms. Multi-version tasks have equivalent functional behaviour (i.e. identical input yields equivalent output), but different non-functional behaviour, namely time and energy consumption. As multi-version tasks are required, we can further exploit this and support versions resulting from e.g. different compiler flags and different functionally-equivalent algorithms. The necessity to include multiple task versions further increases the complexity of reducing the overall energy consumption.

Most modern systems allow Dynamic Voltage and Frequency Scaling (DVFS). For each task, for each CPU type or GPU, there is a clock frequency that minimises the energy consumption. A lower frequency leads to longer runtime and thus increases static energy consumption. A higher frequency leads to shorter runtime but the necessary higher voltage increases dynamic energy consumption. This convex behaviour depends on the code. In modern CPUs clock frequency cannot be altered per core but only per core cluster (i.e. voltage island). Hence, we need to pick the best frequency for each voltage island with respect to the different tasks executing on that island. We take advantage of DVFS for different voltage islands to reduce energy consumption of a whole application consisting of multiple tasks.

Heterogeneous platforms, multiple versions, voltage islands and DVFS extend common scheduling challenges: schedulers now need to decide on which computing unit and at what frequency a task (version) should be executed to reduce the overall energy consumption.

We propose an offline heuristic based approach for multi-version scheduling, which: 1) Fully utilises the heterogeneous CPU and accelerators. 2) Takes advantage of per voltage island DVFS. 3) Dynamically adjusts the frequency throughout the application run time. 4) Selects the optimal version of each task with respect to energy consumption of the whole application.

We choose an offline scheduling approach as online scheduling introduces prohibitive overhead such as keeping in memory all different binaries and all version DVFS data. According to the taxonomy proposed by Davis and Burns [11], our approach can be classified as static, partitioned, time-triggered and non-preemptive.

This paper makes the following contributions:

- We define a task model which integrates multiple versions for dependent tasks.
- We propose a very fine grained energy model.
- Our scheduler embraces heterogeneity by incorporating both different CPU types as well as GPU-style accelerators and by actively controlling both CPU and accelerator DVFS.
- We empirically demonstrate that our approach decreases the energy consumption of an application by more than 15% in comparison to an approach accounting for single-version tasks. Moreover, we show that our approach decreases the energy consumption by on average 45.6% in comparison to a state-of-the-art scheduler.
- We establish that our approach generates applications which are close to optimal with respect to energy consumption.
- We empirically show that our energy-aware scheduler predictions are close to actual energy measurements (largest error -15.8%) on an Odroid-XU4 board.

The remainder of the paper is organised as follows: In Section 2, we describe the system model. In Section 3 we describe the heuristic algorithm. Section 4 details the experimental setup. Section 5 demonstrates the viability of our heuristic scheduler in comparison to a single-version approach, a makespan heuristic and a state of the art energy-aware scheduler. Section 6 shows that our heuristic scheduler produces results close to the optimum and that the predicted energy consumption is close to the actual energy consumption. In Section 7 we discuss related approaches before concluding in Section 8.

2 SYSTEM MODEL

In this section we first detail our platform model (Section 2.1), followed by the task model (Section 2.2) and lastly we explain our energy model (Section 2.3).

2.1 Platform Model

Our approach is fully platform-independent and can be applied to a wide range of heterogeneous (embedded) system architectures. Our model supports: multiple voltage islands, DVFS, heterogeneous CPUs, GPU-style accelerators and in application frequency switching. Additionally, we support GPU tasks that do not only require the GPU but also a CPU control core. The only restrictions our approach has is that a voltage island needs to be homogeneous and that tasks can be bound to a specific core.

Our approach does not limit the number of voltage islands. Each voltage island can have a differing number of cores and DVFS parameters. We consider co-processors as voltage islands. Our approach does not limit the number of different co-processors either.

2.2 Task Model

We consider applications represented as Directed Acyclic Graphs (DAG), hereafter called task graphs. In a graph $G = (\tau, E)$ the set of nodes/vertices τ represents the tasks, and the set of edges E represents data dependencies between tasks, i.e. a producer task needs to be completed before the corresponding consumer task may start executing. Our task model supports multiple sources and sinks. Each task graph has a deadline.

Each task τ_i consists of a (non-empty) set of task versions V. The different versions of a task τ_i are functionally equivalent (i.e. they implement the same input/output relation), but differ in their non-functional properties. Different versions can be the result of: (1) targeting different functional units (i.e. big core, LITTLE core or GPU); (2) using varying compilation flags to, for example, optimise code for energy consumption, binary size, speed or architecture features [23]; (3) different algorithms or implementation variants. This results in a large state space, where picking the best version for a given task can be challenging. All tasks in the task graph must be executed. However, only one version of each task is executed. Hence, the scheduler chooses the version to achieve the best trade-off between energy and performance while meeting the given deadline.

Figure 1 presents a synthetic task graph which we use to illustrate our approach. It consists of nine different tasks: one generator task, one storage task and seven computational tasks in between. The seven computational tasks have one or two different versions each. They target different compute unit types: CPU and GPU. Then, edges are labelled and represent data transfer (or data dependencies) between tasks.

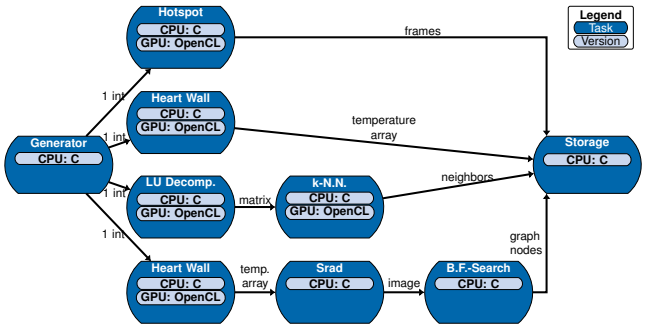

Figure 1: Illustration DAG composed of Rodinia benchmark tasks from Section 6.3 (Task graph number 159).

2.3 Energy Model

Aligned with the state-of-the-art [7, 18, 20] our energy model Equation (1) consists of a static part (E_s) and a dynamic part (E_d). Both static and dynamic energy consumption are computed during scheduling as they depend on scheduling decisions (e.g. selected versions, selected frequencies).

$$E = E_s + E_d \tag{1}$$

In contrast to previous research we consider the impact of frequency on static energy consumption. Frequencies are not continuous and dynamic energy consumption is measured per task

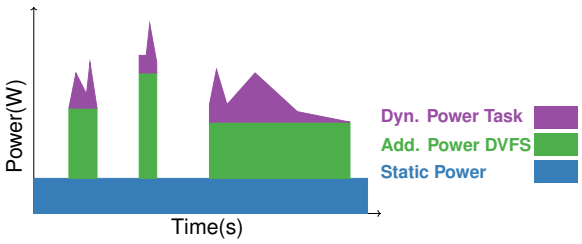

Figure 2: Energy Model

Figure 2 illustrates our energy model based on an example, the x-axis represents the time and the y-axis represents the power. The checkerboard blue box (spanning the complete width of the figure) represents the energy consumption due to the device being powered up. The crosshatch green boxes represent the energy consumption required to run at higher frequencies. The third part of our energy model is the dynamic power required to run a task represented by the irregular shapes on top of the static power components.

$$E_s = E_{sm} + E_{sf} \tag{2}$$

The static energy consumption (E_s), Equation (2), can be split into two elements E_{sm} and E_{sf}. Firstly, E_{sm} corresponds to the energy consumed by the board because it is powered on and depends only on the worst-case duration (i.e. application makespan) it remains powered on. Equation (3) multiplies the average measured power in Watt ($W_{average}$) required at the lowest frequency by the overall schedule makespan (C) of the application.

$$E_{sm} = C * W_{average} \tag{3}$$

Secondly, E_{sf} corresponds to the additional energy consumption required to operate the board at a given frequency. As the board consists of multiple voltage islands, the E_{sf} (Equation (4)) corresponds to the consumed energy by all voltage islands ($i \in I$) at each specific frequency ($f \in F_i$).

$$E_{sf} = \sum_{i \in I} \sum_{f \in F_i} C_{i,f} * W_{i,f} \tag{4}$$

Hence, the time spent on each voltage island at each frequency ($C_{i,f}$) is multiplied by the average additional energy consumption at that frequency ($W_{i,f}$). This time depends on the schedulers decisions and more precisely on the worst-case execution time (WCET) of the selected version for each task.

It is important to not over-accumulate the time spent in each frequency. The WCET of two tasks executing concurrently on the same voltage island should not be summed up to compute the time spent in a given frequency. Instead it is the longest time of the two tasks that must be accounted for. If two tasks are executed on different voltage islands the WCET of each task must be accounted for, as the time spent in each frequency is voltage island specific.

The dynamic energy consumption, Equation (5), is dependent on the workload executed by a compute unit at a given frequency. Unlike most existing power models, we measure the energy consumed by each task version for each frequency on each corresponding compute unit. Measures are performed *a-priori* to scheduling the application and in isolation (all other compute units are idle, and no other task is executing). To compute the dynamic energy consumption we measure the total energy consumption and then subtract the two static power components (i.e. the energy consumption due to running at the base DVFS and the additional energy consumption due to the increased frequency). That is also the reason why we work with average power ($W_{average}$) for the static part, as using the worst measured idle power could lead to a negative dynamic energy consumption for some tasks.

The total dynamic energy, Equation (5), consumed by an application E_d is the sum of all selected task versions dynamic energy at a given frequency targeting a specific compute unit ($E_{p,u,f}$). This approach allows to better account for different energy requirements of tasks since Balsini et al. [5] and Vasilakis et al. [30] showed that a one-size-fits all dynamic energy consumption for the whole application is unrealistic.

$$E_d = \sum_{p \in \tau} E_{p,u,f} \tag{5}$$

Not all task versions are present in the final energy consumption estimation. The version selection depends on the scheduling decisions. Even-though we skipped this constraint in above equations for clarity, it is present in scheduling Algorithm 3.2. Splitting both static and dynamic energy consumption allows us to model DVFS for the three voltage islands on the Odroid-XU4 platform. This concept can be extended to account for all additional DVFS capable compute units or voltage islands.

Similar to Guo et. al [18] we neither consider Dynamic Power Management (DPM) nor the switching cost of changing frequency of the processors. DPM is only beneficial when the idle slot is longer

than a certain threshold. The idle time might not be long enough as all CPU cores in a cluster have to be idle. Additionally, we already decrease the frequency to the least possible if all cores in a cluster are idle. We do not consider the frequency switching cost as it is comparable to the cost of context switches in a multitasking environment (i.e. marginal) [24, 27].

3 ENERGY-AWARE FORWARD LIST SCHEDULING

Our proposed heuristic is based on Forward List Scheduling (FLS). FLS first orders the tasks, then adds them one by one to the schedule without backtracking. We use two sorting algorithms: Depth First Search (DFS) and Breadth First Search (BFS). For both sorting strategies, we use the task WCET as a tie breaking rule (larger WCET to be scheduled first). Furthermore, we introduce two additional tie breaking rules for BFS, one based on *time laxity* and one based on *energy laxity*. Since it is shown in [25] that no sorting algorithm consistently outperforms the others, we generate four schedules, each resulting from one sorting strategy / tie breaking rule combination, and select the one resulting in the lowest energy estimation as our heuristic solution. From here on-wards we will refer to our energy-aware, multi-version task scheduling heuristic as eFLS.

ALGORITHM 3.1: Scheduling algorithm

```
Input  : An application DAG composed of multi-version tasks (τ), a
         set of computational units (CU) and a deadline (D^C).
Output : A schedule.
1  Function ListSchedule(τ = τ_1, ..., τ_n | τ_x = (v), CU, D^C)
2      Qready ← Topological_Sort_Tasks(τ)
3      Qdone ← []
4      schedule ← new Schedule()
5      while t ← Qready.pop_front() do
           // tmpSched best schedule for the current task
6          tmpSched ← new Schedule()
7          tmpSched.energy ← ∞
8          foreach v ∈ t.versions do
9              foreach u ∈ CU do
10                 if v runs on u then
11                     copy ← schedule
12                     copy.Schedule_task(Qdone, t, v, u)
13                     copy.Update_energy()
14                     if copy.energy < tmpSched.energy then
15                         tmpSched ← copy
16         schedule ← tmpSched
17         schedule.Update_makespan()
18         if schedule.makespan > D^C then
19             return unschedulable
20         Qdone.push_back(t)
21     return schedule
```

Our proposed scheduling algorithm is sketched out in Algorithm 3.1. It uses the task graph, a list of computational units (CU) and the application deadline (D^C) as inputs. First it sorts the tasks to be scheduled and creates a list (Line 2). Then, a loop iterates over all tasks while there exist tasks to be scheduled (Lines 5–20). Each tasks has multiple versions that are all tested on all possible compute units (Lines 8–9). The different task versions also account for different frequencies, i.e the frequency is a characteristic of v (Line 8). Line 10 checks if a given version can be executed on the given compute unit (u). After scheduling the specific task version to an appropriate compute unit (Line 12, calling Algorithm 3.2), we compute the energy consumption of the new schedule (Line 13). The energy consumption computation includes the two static energy consumption components of the schedule and the dynamic energy consumption of each task. The static base energy consumption and the dynamic energy consumption are straightforward to compute, but the time spent in each frequency on each voltage island is not. We generate a list for each frequency in each compute unit. Then if a task is scheduled on a compute unit we append the WCET of the task to the matching frequency list entry. Next we merge all WCETs for each frequency list if they overlap. This results in a list of time blocks for each frequency and compute unit. Based on the list we can compute the time spent in each frequency, which in turn can then be used for computing the second component of the static energy consumption.

ALGORITHM 3.2: Scheduling of a task

```
Input  : List of scheduled tasks (Qdone), Current task to schedule
         (cur_task) and its version (version), Current processor
         (cur_cu).
Output : Add a new task to the schedule.
1  Function Schedule_task(Qdone, cur_task, version, cur_cu)
2      cur_task.ρ ← max_{x∈predecessors(cur_task)} (x.ρ + x.C)
3      Change ← True
4      while Change do
5          Change ← False
6          foreach t ∈ Qdone do
7              if t is mapped on core cur_cu then
8                  if t overlaps in time with cur_task then
9                      cur_task.ρ ← t.ρ + t.C
10                     Change ← True
11         foreach t ∈ Qdone do
12             if t is mapped on cur_cu.volt_island then
13                 if t overlaps in time with cur_task then
14                     if t.freq ≠ version.freq then
15                         cur_task.ρ ← t.ρ + t.C
16                         Change ← True
17     cur_task.update(version)
18     add_task(cur_task)
```

The version and mapping resulting in the lowest energy estimation is selected (Lines 14–15). Thus the selection of the best compute unit and version is greedy. In Line 18 we check if the overall schedule makespan is less than the deadline (D^C). And finally we add the scheduled task to the list of scheduled tasks *Qdone* (Line 20). When all tasks are scheduled the final schedule is returned (Line 21).

Scheduling a task. Algorithm 3.2 sketches out the method to determine the start time of the current task (*cur_task*). Our approach uses an *As Soon As Possible* (ASAP) strategy. Each task must start after its causal predecessors finished (Line 2), where ρ is the start time of a task and C is the runtime of a task. Then, while there have been changes (Line 4) in the last iteration we enforce: 1) that there is no overlap between two tasks that require the same compute unit (Lines 6 – 10); 2) that all tasks running at the same time on the same voltage island run at the same frequency (Lines 11 – 16). If any of these two cases happens, the start time (ρ) of the current task (*cur_task*) is postponed (Lines 9 and 15). Postponing the start of the current task might not be optimal in the case of non-matching frequencies. However, we test more than one frequency. If the current task can be executed at the frequency of the other tasks this scenario is also explored. The algorithm compares the different alternatives and selects the best. Lastly, the task member attributes are updated with the version attributes (Line 17) and we add the task to the schedule (Line 18).

Algorithm 3.2 is guaranteed to terminate because we only postpone the current task, which can be moved as far as the end of the current schedule. In this case, this task would be executed without any other concurrent task. This ensures that all `if`-conditions are satisfied and no more changes would be required.

4 EXPERIMENTAL SETUP

In Section 4.1 we describe the target hardware. Followed by Section 4.2, which details our energy measurement setup. Then we introduce the task graph generation (Section 4.3) that we use throughout Sections 5 and 6. At last, we explain our DVFS approach for the target platform (Section 4.4).

4.1 Target Platform

Our approach can target a wide variety of target platforms, as described in Section 2. However, for the sake of illustration and concrete experimental validation we focus on the Odroid-XU4 board from here onward.

The Odroid-XU4 [2] platform is based on the ARM big.LITTLE CPU architecture [4] complemented by a Mali GPU accelerator [3]. The CPU is an Exynos 5 Octa 5422 chip [3], which embeds two clusters of four cores each. One cluster includes energy efficient in-order Cortex-A7 cores (LITTLE) while the other includes high-performance, out-of-order, deep-pipeline Cortex A-15 cores (big). Each cluster forms a separate voltage island, i.e. the voltage and frequency can only be changed for all cores in a cluster at the same time. The two core types are ISA-compatible. Each core has its own L1 cache, and each cluster has a shared L2 cache (512KB for LITTLE cores, 2MB for big cores). The Mali GPU features 6 shader cores and shares the off-chip physical memory with the CPU, thereby avoiding common data transfer overhead between CPU and GPU. The GPU is a third voltage island independent from the CPU.

4.2 Energy Measurements

Energy consumption is measured with the Otii system by Qoitech[1]. It is a non-intrusive high-side power monitor with a sampling rate of up to 4kHz. Our target Odroid-XU4 platform communicates the start of software tasks via UART to the Otii at 115200bps. To ensure fair measurements we take the following steps:

- We do not take into account the energy consumption of the fan, which is powered via a separate device.
- We calibrate the Otii device before the experiments.
- We warm up the involved devices by executing tasks before the actual experiments.
- We measure time (WCET), power and energy to solution (EtoS) at the same time.
- We run each experiment 50 times, to obtain enough data for statistical testing.

[1] https://www.qoitech.com/products/standard

4.3 Application code

In order to compare different scheduling methods we generate random DAG based applications. To build the structure of the graph we rely on Task Graphs For Free (*TGFF*) [14]. Task graphs are generated with a total of eight different types of tasks. Then we *a-posteriori* perform a mapping between *TGFF* tasks (nodes) to Rodinia benchmark tasks [9]. This results in a large collection of randomly generated task graphs with executable code.

Each task includes all IO, overhead and computations required for the Rodinia benchmark. We work with the following 8 benchmark tasks: *heartwall*, *hotspot*, *k-means*, *lud*, *nn*, *nw*, *bfs* and *srad_v1*. We use these benchmarks because only these could be executed on the Odroid-XU4 with minor changes. These changes among others ensure thread safety when multiple instances of the same benchmark are run concurrently. The Rodinia benchmark suit does not provide a sequential implementation of the benchmarks. Therefore, we use the OpenMP implementations and execute them on a single thread. Additionally, we use the OpenCL implementations of six of these benchmarks to demonstrate multi-version task scheduling. A similar approach was taken by De Bock et. al. [12], who used TACLe benchmarks [15] for independent tasks.

The energy and timing information required for our approach can be obtained with different methods such as measurements or static analysis. In this paper we chose to obtain the required information through measurements. Static analysis is not possible as the target architecture is non-deterministic. The dynamic energy consumption and timing measurements were collected at the same time and in isolation for each task. The measurements cover all possible execution paths of the tasks and we selected the worst observed values as the WCET estimates. To further increase the safety of our estimations and to account for contention, we increased the observed WCET by an arbitrary chosen safety margin of 30%.

4.4 DVFS

Previous papers [7, 10, 13] have shown that the energy consumption of an application with respect to the frequency follows a convex curve, i.e. the lowest energy consumption of an application is achieved at a mid-level frequency. A lower frequency results in long run times and, therefore, in a higher energy consumption. A higher frequency results in a shorter run time, however the voltage increases required for the frequency increase offsets the shorter run time [7, 10, 13].

On the big cores we find similar convex behaviour for all selected Rodinia benchmarks. The minimum energy consumption is always achieved between 1.3GHz and 1.6GHz. Therefore, we consider all frequencies between 1.3GHz and 2GHz (i.e. the maximum clock frequency possible), allowing for a good trade-off between energy consumption and run time. On the LITTLE cores the convex behaviour is not as pronounced as on the big cores (i.e. increasing the frequency from 1.3GHz to 1.4GHz does not increase the energy consumption by much). The minimum energy consumption is achieved between 1.3GHz and 1.5GHz (i.e. the maximum clock frequency possible). Hence, we consider all frequencies between 1.3GHz and 1.5GHz. For the GPU we consider all supported frequencies.

5 eFLS COMPARISONS

To validate our proposed approach we conduct a series of experiments. For this purpose we generate 500 task graphs with TGFF [14], with an average of 124.8 tasks. All experiments in Section 5 are based on the same task graphs.

First, Section 5.1 compares a multi-version task eFLS approach to a single-version task eFLS approach. Then, we contrast our eFLS heuristic to a multi-version task strategy based on classic Forward List Scheduling (FLS) (Section 5.2). At last, we scrutinise if the eFLS heuristic performs as well as a state-of-the-art meta-heuristic based scheduler (Section 5.3).

5.1 Single-version vs. Multi-version tasks

We compare our multi-version task eFLS scheduler against a single-version task eFLS scheduler. The single-version task scheduler is the same as the multi-version task scheduler, however, it has access to only the CPU version or to the GPU version of a task.

If the single-version eFLS scheduler only has access to the CPU versions, then the multi-version task eFLS solutions are on average 15.7% more energy efficient and take less time than the single-version task eFLS solutions. The multi-version task solutions are up to 35.6% more energy efficient and are at least 6.9% more energy efficient. The histogram of energy consumption reductions is shown in Figure 3.

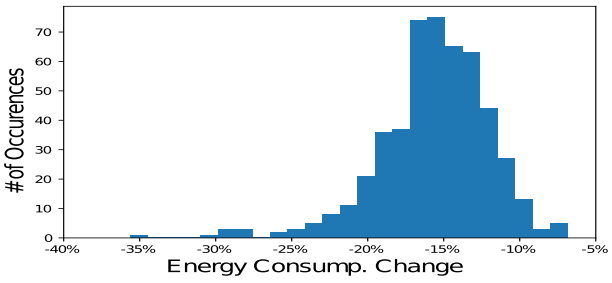

Figure 3: Histogram of the energy consumption reductions when providing multiple versions to the eFLS scheduler.

These results clearly demonstrate that including the GPU versions is beneficial with respect to energy consumption. One might argue that instead of using the CPU versions for all tasks, using only the GPU version for some tasks will be more beneficial. However, it is not clear before scheduling which task should be executed on the GPU as this differs per task graph. For example, the *k-means* tasks are scheduled on the GPU in 24.2% of the cases and the *LU decomposition* tasks are scheduled on the GPU in 77.0% of the cases. Combining this knowledge with the fact that there is only one GPU (i.e. GPU tasks have to run sequentially after each other), it is not surprising that providing the *k-means* and *LU decomposition* GPU versions to the single-version eFLS scheduler does not improve the situation. Our results show that the multi-version schedules are on average 11.3% more energy efficient than the single version approach with the *k-means* and *LU decomposition* GPU versions. Thus, including multiple versions and letting the scheduler pick the best one has clear advantages. Decreasing energy consumption by more than 15% could be a real game changer for battery operated devices.

5.2 Energy optimising vs. Makespan

Next we compare our multi-version task eFLS scheduler against a multi-version task energy-unaware scheduler. We use the same FLS approach as introduced in Section 3, however instead of selecting the best version based on energy consumption, we select the best version based on run time. This way we mimic existing methods, e.g. [25].

As the FLS schedule focuses purely on minimising makespan, it takes full advantage of the high clock speeds available on the Odroid-XU4, thereby increasing the voltage to the maximum. Thus, it is not surprising that the eFLS generated schedules consume on average 25.1% less energy than the FLS generated schedules, with a standard deviation of 3.4%. The eFLS solutions are always more energy efficient. They are at least 11.4% more energy efficient and at most 33.3%. The makespan focus (and thus high frequency) means that the FLS solutions result in a makespan which is on average 19.3% lower than the makespan of the eFLS solutions. Thus, we can show that our approach to minimising energy consumption is valid.

5.3 eFLS vs. ARSH-FATI

As shown by Sheikh and Pasha [27], heterogeneous energy efficient scheduling methods so far have mostly focused on optimising the energy consumption of multiple independent tasks. A paper published by Ullah Tariq et. al. [29] is the only exception that we are aware of. The authors researched energy-efficient static scheduling for DAG task graphs with deadline constraints. The paper focuses on Smart Networked Systems, and experimental results are simulation-based. The paper explores scheduling, mapping and DVFS based on population heuristics. The solutions generated by their approach are 24% more energy efficient than CA-TMES-Search and 30% more energy efficiency than CA-TMES-Quick [19].

Their approach only selects the per voltage island DVFS once, whereas our approach can switch between frequency levels at runtime. Additionally, the approach by Ullah Tariq et. al. neither differentiates between task versions, nor takes into account the GPU. We implemented the proposed meta-heuristic from the description in

their paper and used it to schedule the same task graphs as in Sections 5.1 and 5.2. Besides the population size we used the same hyper parameters as determined in [29] ($DR = 0.3$, $\lambda = 0.9$,$NI = 500$). The population size in the original paper was limited to 5, increasing the population size to 500 improved the performance of the meta-heuristic significantly.

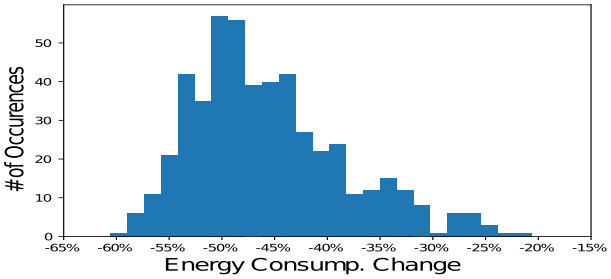

Figure 4: Histogram of the energy consumption reductions of our eFLS approach over the ARSH-FATI meta-heuristic.

Figure 4 shows the improvement of our approach over the ARSH-FATI meta-heuristic. Our approach produces schedules that are on average 45.6% more energy efficient, at most 60.6% more efficient and at least 20.7% more energy efficient.

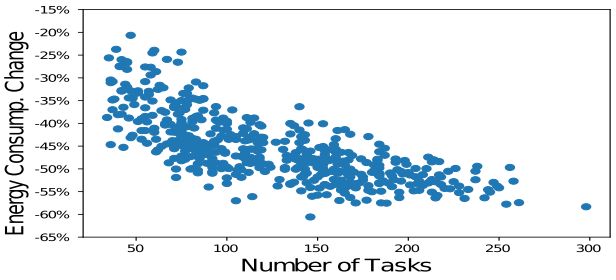

Figure 5: Energy consumption reduction of eFLS vs ARSH-FATI with respect to the number of tasks in a taskgraph.

Further analysis indicates that ARSH-FATI performs better on smaller task-graphs as shown in Figure 5. An additional experiment with smaller taskgraphs (average of 9.78 tasks) shows that the gap between ARSH-FATI and eFLS is indeed smaller. However, the eFLS scheduler still produces solutions that are on average 21.9% more energy efficient.

The significant difference between eFLS and ARSH-FATI shows that our approach can explore the optimisation state-space better than the current state-of-the-art approach.

In conclusion our experiments show that: multi-version outperforms single-version, our approach to energy scheduling is valid and our approach is better at handling the optimisation space than the current state-of-the-art approach.

6 eFLS VS. OPTIMAL

Heuristic algorithms return approximate solutions by nature. Determining the over-approximation ratio requires to compare the results of the heuristic with those of an exact method. Fortunately, for scheduling problems it is possible to generate optimal solutions using an Integer Linear Programming (ILP) formulation. However, solving ILP scheduling problems is an NP-hard problem and thus does not scale well with increasing number of tasks. To estimate the over-approximation of our eFLS heuristic we also introduce an ILP-based scheduler, which is summarised in Section 6.1. Then we compare the ILP generated solutions to the eFLS generated solutions in Section 6.2. At last, we show that the predicted energy consumption matches the measured energy consumption (Section 6.3).

6.1 ILP formulation

An ILP formulation consists of a set of constraints that must be satisfied and an objective function that will be optimised. In some of the following constraints, logical operators $\vee$ and $\wedge$ are used for clarity, these operators can be linearised using [8].

Objective function. Our goal is to minimise the energy consumption over all tasks of an application as formalised by Equation (6). The energy consumption E of a schedule is equal to the sum of the static energy consumption and the dynamic energy consumption, as stated by Equation (1) in our energy model.

Map a task to a compute unit. Equation (9) ensures that task p is mapped on one and only one compute unit u ($m_{p,u} = 1$). Equation (10) indicates if two tasks, p and q, are assigned to the same compute unit $s_{p,q} = 1$.

$$minimise\ E = makespan \times W_{average} \tag{6}$$

$$+ \sum_{i \in I} \sum_{f \in F_i} acctime_{f,i} \times W_{i,f} \tag{7}$$

$$+ \sum_{p \in \tau} E_p \tag{8}$$

$$\sum_{u \in CU} m_{p,u} = 1, \forall p \in \tau \tag{9}$$

$$s_{p,q} = \sum_{u \in CU} m_{p,u} \wedge m_{q,u}, \forall (p,q) \in (\tau \times \tau), p < q \tag{10}$$

Prevent overlap on same compute unit. If two tasks are mapped to the same compute unit, variable o determines the order of tasks p and q, $o_{p,q} = 1$ means p is scheduled before q. Thus, Equation (11) enforces that two tasks are executed in a given order, and only one of the two orders is possible at once.

Equation (12) prevents time-wise overlap of two tasks on the same compute unit, i.e. q must start after the completion of p, if p is scheduled before q. It uses a big-M nullification [17] to *deactivate* the constraint if tasks are scheduled in the opposite order. M must always be greater than the left-hand side of the equality, we therefore use the sequential makespan of the application $M = \sum_{p \in \tau} max(C_p)$, as many other papers, e.g. [25].

$$s_{p,q} = o_{p,q} + o_{q,p}, \forall (p,q) \in (\tau \times \tau), p < q \tag{11}$$

$$\rho_p + C_p \leqslant \rho_q + (1 - o_{p,q}) \times M, \tag{12}$$
$$\forall (p,q) \in (\tau \times \tau), p \neq q$$

Data dependencies in task graphs. Equation (13) ensures that if one task p depends on the data of another task q, the start time of p (ρ_p) is greater than the end time of q ($\rho_q + C_q$).

$$\rho_p \geqslant \rho_q + C_q, \forall p \in \tau, \forall q \in predecessors(p) \tag{13}$$

Task version selection. Equation (14) enforces that exactly one version of each task is selected ($a_{p,i} = 1$). Each version is mapped to one and only one core ($x_{p,i,m} = 1$), Equation (15). And Equation (16) links version and accepted architecture ($x_{p,i,m}$). Then, Equation (17) sets the selected mapping at the task level ($w_{p,m}$).

$$\sum_{v \in v_p} a_{p,v} = 1, \forall p \in \tau \tag{14}$$

$$a_{p,v} = \sum_{u \in CU} x_{p,v,u}, \forall p \in \tau, \forall v \in v_p \tag{15}$$

$$x_{p,v,u} = 0, \forall p \in \tau, \forall v \in v_p, \forall u \in forbidden(v) \tag{16}$$

$$w_{p,u} = \sum_{\forall v \in v_p} x_{p,v,u}, \forall p \in \tau, \forall u \in CU \tag{17}$$

Energy & Timing & Frequency. Equations (18) to (20) set the energy, time, and frequency for each task (E_p, C_p, F_p) to the energy, time and frequency ($E_{p,v,u}$,$C_{p,v,u}$,$F_{p,v,u}$) of selected version $x_{p,v,u} = 1$.

$$E_p = \sum_{v \in v_p} (x_{p,v,u} \times E_{p,v,u}), \forall p \in \tau, \forall u \in CU \tag{18}$$

$$C_p = \sum_{v \in v_p} (x_{p,v,u} \times C_{p,v,u}), \forall p \in \tau, \forall u \in CU \tag{19}$$

$$F_p = \sum_{v \in v_p} (x_{p,v,u} \times F_{p,v,u}), \forall p \in \tau, \forall u \in CU \tag{20}$$

Timing constraints. Equation (21) guarantees that all sink nodes (s) of an application DAG complete execution ($\rho_s + C_s$) before the (global) deadline D.

$$\rho_s + C_s \leqslant D, \forall s \in sinks(\tau) \tag{21}$$

Consistent voltage island. When multiple tasks are mapped on the same voltage island at the same time, their frequencies must match as required by a voltage island. Equation (22) sets on which island i the task p is mapped ($is_{p,i} = 1$), while Equation (23) checks if two tasks p, q are on the same island ($sis_{p,q} = 1$). Then, Equation (24) checks if two tasks p, q overlap in time ($to_{p,q} = 1$) (obviously on different cores as enforced by previous constraints Equation (11)). And Equation (25) forces the frequency of two tasks p, q to be equal, $F_p = F_q$, if they are on the same voltage island at the same time.

$$is_{p,i} = \sum_{u \in i} (w_{p,u}), \forall p \in \tau, \forall i \in I \tag{22}$$

$$sis_{p,q} = \sum_{i \in I} (is_{p,i} \wedge is_{q,i}), \forall (p,q) \in (\tau \times \tau), p < q \tag{23}$$

$$to_{p,q} = (\rho_q + C_q) \geq \rho_p \wedge (\rho_p + C_p) \geq \rho_q, \forall (p,q) \in (\tau \times \tau), p < q \tag{24}$$

$$(sis_{p,q} \wedge to_{p,q}) \times (F_p - F_q) = 0, \forall (p,q) \in (\tau \times \tau), p < q \tag{25}$$

Time spent in each CPU frequency. To compute the time spent by each voltage island at each frequency, we must look at each time quantum if there is a task active. Equation (26) scans all time steps between 0 and M, which is the longest possible (sequential) schedule, then set $ta_{t,p} = 1$ if the task p is active at that time. Note that it would be better to use the makespan of the schedule rather than M, but the makespan results from scheduler decision and is therefore unknown when modelling the problem. Equation (27) then sets the binary variable $fa_{i,f,t} = 1$ if at least one task is active at time t with frequency f on island i. Finally, Equation (28) accumulates the time at which the island i runs at frequency f.

$$ta_{t,p} = (t \geqslant \rho_p) \wedge (t \leqslant (\rho_p + C_p)), \forall t \in [0; M], \forall p \in \tau \tag{26}$$

$$fa_{i,f,t} = ta_{p,t} \wedge (F_p == f) \wedge is_{p,i}, \forall t \in [0; M], \forall i \in I, \forall f \in F_i, \forall p \in \tau \tag{27}$$

$$acctime_{i,f} = \sum_{t \in [0;M]} fa_{i,f,t}, \forall i \in I, \forall f \in F_i \tag{28}$$

6.2 ILP vs. eFLS

To estimate the over-approximation of our eFLS heuristic (Section 3) over the optimal solution we generate 500 task graphs with TGFF and schedule them with both techniques. We then compare the predicted energy consumption of the generated schedules, i.e. we calculate the expected energy consumption of the task graph with respect to the two schedules and compare the energy consumption. On average the generated task graphs have 9.78 tasks with a standard deviation of 4.53.

For each technique, the solving time vs. amount of tasks can be found in Figure 6. The figure clearly shows that the ILP solving time increases exponentially with the number of tasks. Hence, demonstrating that the ILP does not scale well.

The ILP approach found solutions with a gap smaller than 0.1% for 45% of the task graphs within 24 hours on a 16 core Intel Xeon Gold 6130 using Cplex. For the remaining 55% the solver found solutions with a gap larger than 0.1% in 38.2% of the cases and no solution in the last 16.8% of the task graphs.

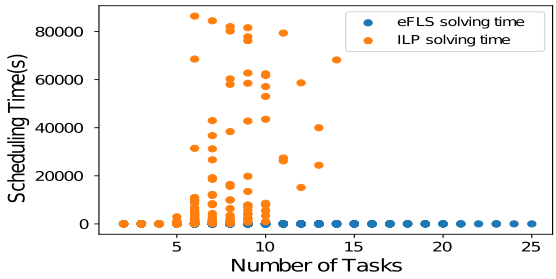

Figure 6: Solving time (s) of the ILP solver vs eFLS heuristic

The average energy degradation of the eFLS solution, with respect to the solved ILPs, is 1.6%, which we deem an acceptable trade-off for shorter scheduling times and better scalability. The degradation distribution is shown in Figure 7. At best both methods result in the same schedule (in 24.2% of the cases). At worst the eFLS method results in a schedule that consumes 15.9% more energy.

6.3 Energy consumption: predicted vs measured

In this section we compare the predicted energy consumption from the scheduler to the actual energy consumption when executed on the Odroid-XU4 board. There is no significant difference between the predicted makespan and the actual makespan as we employ a time-triggered approach.

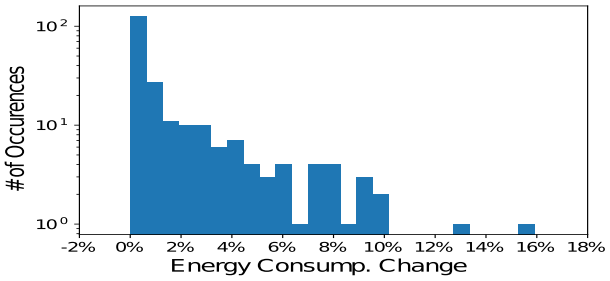

Figure 7: Distribution of energy degradation of the eFLS scheduler vs the ILP scheduler. (logarithmic scale)

We choose five arbitrary task graphs (Table 1) from Section 6.2 and execute the applications, for both the ILP and the eFLS generated schedules, 50 times on the Odroid-XU4. An example task graph is shown in Figure 1. It consists of 9 tasks, where 5 of the tasks have GPU versions.

For the task graph in Figure 1 both the ILP and the eFLS scheduler make use of the GPU for the *LU decomposition* task and execute the other tasks on the CPU. The largest difference between the two schedules is that the ILP generated schedule mostly executes tasks at 1.3GHz and two tasks at 1.5GHz, whereas the eFLS schedule executes most tasks at 1.5GHz. Therefore, the eFLS generated schedule is a little bit faster and slightly worse at balancing runtime and energy consumption. For sake of brevity we do not discuss the other task graphs in detail.

Table 1: The maximum and average error of the measured energy consumption vs the predicted energy consumption for each selected task graph, for both the ILP and the eFLS solutions.

Task Gr.	#Tasks	ILP		eFLS		Degradation	
		max. er.	avg. er.	max. er.	avg. er.	Pred.	Meas.
97	6	-15.6%	-15.2%	-15.2%	-14.6%	7.9%	8.7%
151	7	-12.7%	-12.2%	-14.6%	-14.0%	3.0%	1.0%
159 [2]	9	-14.8%	-14.3%	-14.7%	-14.2%	2.7%	2.8%
225	10	-15.6%	-15.2%	-15.8%	-15.4%	3.0%	2.7%
320	6	-9.0%	-9.0%	-9.0%	-9.0%	0.0%	0.03%

Table 1 shows the error between the measured and the predicted energy consumption. All predictions over-estimate the energy consumption, which is likely due to the safety margin. The predicted energy consumption is at most off by 15.8% and at least by 9.0%. The measured energy consumption degradation of the eFLS solution is almost the same as the degradation of the predicted energy consumption. Thus, our predictions can be used to compare scheduling methods.

7 RELATED WORK

The majority of research until 2016 focused on energy-efficient scheduling for homogeneous multi-core platforms, for surveys see [6, 16]. The two most explored techniques were Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM). DVFS techniques focus on balancing clock frequency and voltage with required system performance, aiming for the best trade-off between energy consumption and execution time. DPM techniques switch parts of the CPU into a low-power state, thereby reducing energy consumption [6, 16].

In recent years attention has shifted from homogeneous to heterogeneous systems. Scheduling techniques now account for core mapping and energy efficiency of tasks [26].

Most research in the hard-real time community focuses on two-type cores [21] and on scheduling independent tasks [26, 28]. In contrast, we address dependent tasks for multi-type CPUs with on-board GPU. Additionally, we are not aware of any energy minimising approaches that work with multi-version tasks.

Most previous publications use energy models that rely on a standard power model that estimates the power consumption based on a static and a dynamic part [7, 18, 20, 28, 29]. However, this ignores that different tasks may consume very different amounts of dynamic energy even if the tasks take the same amount of time (e.g. integer division vs double multiplication on big cores). Our energy model is based on per task measurements. Thus, switching to a different architecture does not require a new power model, but merely re-measuring tasks. Additionally, we do not consider core frequencies to be continuous, as this does not reflect the actual hardware. Instead we consider the true set of available frequencies. Lastly, previous research does not take into consideration DVFS for the static energy consumption.

Further differences to previous work are:

Unlike Guo et. al [18] we do not only focus on CPU power consumption but also on GPU power. Additionally, we work with time-triggered dependent tasks instead of independent parallel sporadic tasks.

Zahaf et. al. [31] proposed scheduling approaches for soft real-time tasks running on heterogeneous multi-core platforms. They introduce integer non-linear programming (INLP) and heuristics to determine the best amount of parallelism for each independent task. In comparison, we minimise the energy consumption of dependent tasks, which are executed concurrently.

Thammawichai and Kerrigan [28] work with two-type heterogeneous multiprocessors, focusing on independent, preemptive tasks. The three approaches introduced use mixed-integer nonlinear program (MINLP), non-linear programming and a task ordering algorithm. Our work focuses on dependent non-preemptive dependent tasks.

To summarise, in comparison to previous research, our work focuses on reducing energy consumption for dependent, time-triggered, multi-version tasks on single heterogeneous devices. Furthermore, we extend the aspect of heterogeneity to not only take into account a heterogeneous CPU, but also the GPU. We introduce a more realistic energy model. Lastly, our experimentation is based on the Odroid-XU4, a more modern board than used in previous research.

[2] Shown in Figure 1

8 CONCLUSION

High-performance embedded systems such as the Odroid-XU4 or the Nvidia Jetson boards are becoming ubiquitous. Many application areas requiring such platforms are battery powered, hence software executing on them needs to consume as little energy as possible. To achieve this we need to take into account all features of modern hardware. To the best of our knowledge we are the first to propose a scheduling approach that combines heterogeneous CPU and GPU, multi-version dependent tasks, a fine grained energy model and in application frequency switching.

Our eFLS scheduling heuristic embraces heterogeneity and incorporates both different CPU types as well as GPU-style accelerators. The scheduling heuristic can not only change the frequency at regular intervals in the application but also employs an energy model to steer its direction that is more fine-grained then previous research energy model. Lastly, it scales well with the size of task-graphs.

We demonstrate that our novel multi-version scheduling approach can take full advantage of a heterogeneous system, reducing energy consumption by more than 15% over a single version approach. We also show that there is not one single best version for all tasks across all task graphs. This provides evidence that the multi-version approach produces more energy-efficient schedules than a single-version approach. We show that our eFLS scheduling approach outperforms a standard FLS scheduler and the state-of-the-art ARSH-FATI [29] scheduler by on average 45.6% with respect to energy consumption.

Our energy-aware Forward List Scheduling (eFLS) solutions experience a mean degradation of only 1.6%, with respect to energy consumption in comparison to the optimal Integer Linear Programming (ILP) solutions. Lastly, we show that the energy predictions of our scheduling approach is similar to the measured energy consumption with a maximum error of 15.8%.

We improve on the state-of-the-art scheduling method and show that our approach can explore the optimisation state-space better. We show that a multi-version approach has advantages over a single-version approach. Lastly, the solutions generated by our heuristic are very close to optimal.

In the future we plan to extend our scheduling approach to take into account security, thereby allowing for different trade-offs between time, energy and security. Furthermore, we plan to extend our approach to incorporate hardware resilience and thereby be more applicable to critical cyber physical systems.

ACKNOWLEDGMENT

We would like to thank the reviewers for their time and feedback.

This work is supported and partly funded by the European Union Horizon-2020 research and innovation programme under grant agreement No. 779882 (TeamPlay).

REFERENCES

[1] [n. d.]. Nvidia Jetson. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/. Accessed: 2019-09-06.
[2] [n. d.]. Odroid-XU4. https://wiki.odroid.com/odroid-xu4/odroid-xu4. Accessed: 2019-09-06.
[3] 2019. Exynos 5 Octa 5422 Processor: Specs, Features: Samsung Exynos. https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-octa-5422/
[4] ARM Ltd. 2013. White Paper: big.LITTLE Technology : The Future of Mobile. (2013).
[5] A. Balsini, L. Pannocchi, and T. Cucinotta. 2018. Modeling and simulation of power consumption and execution times for real-time tasks on embedded heterogeneous architectures. In EWiLi 2018.
[6] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo. 2016. Energy-Aware Scheduling for Real-Time Systems. TECS 15, 1 (2016).
[7] A. Bhuiyan, Z. Guo, A. Saifullah, N. Guan, and H. Xiong. 2018. Energy-efficient real-time scheduling of dag tasks. TECS 17, 5 (2018).
[8] G.G. Brown and R.F. Dell. 2007. Formulating integer linear programs: A rogues' gallery. ITE 7, 2 (2007).
[9] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, S. Lee, and K. Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In IISWC. IEEE.
[10] A. Colin, A. Kandhalu, and R. Rajkumar. 2016. Energy-Efficient Allocation of Real-Time Applications onto Single-ISA Heterogeneous Multi-Core Processors. J. of Signal Processing Systems 84, 1 (2016).
[11] R.I. Davis and A. Burns. 2011. A survey of hard real-time scheduling algorithms for multiprocessor systems. Comput. Surveys (2011).
[12] Y. De Bock, S. Altmeyer, T. Huybrechts, J. Broeckhove, and P. Hellinckx. 2018. Task-set generator for schedulability analysis using the TACLeBench benchmark suite. ACM SIGBED Review 15, 1 (2018).
[13] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho. 2013. The energy/frequency convexity rule: Modeling and experimental validation on mobile devices. In PPAM. Springer, 793–803.
[14] R.P. Dick, D.L. Rhodes, and W. Wolf. 1998. TGFF: task graphs for free. In 6th CODES/CASHE. IEEE.
[15] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R.B. Sørensen, P. Wägemann, and S. Wegener. 2016. TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research. In 16th WCET, Vol. 55. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
[16] M.E.T. Gerards, J.L. Hurink, and P.K.F. Hölzenspies. 2016. A survey of offline algorithms for energy minimization under deadline constraints. J. of Scheduling 19, 1 (2016).
[17] I. Griva, S.G. Nash, and A. Sofer. 2009. Linear and nonlinear optimization. Vol. 108. Siam.
[18] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N Guan. 2019. Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms. RTAS 2019-April (2019).
[19] Jian-Jun Han, Man Lin, Dakai Zhu, and Laurence T Yang. 2014. Contention-aware energy management scheme for NoC-based multicore real-time systems. TPDS 26, 3 (2014), 691–701.
[20] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele. 2014. Energy efficient DVFS scheduling for mixed-criticality systems. EMSOFT 354 (2014).
[21] O. Khan and S. Kundu. 2010. A self-adaptive scheduler for asymmetric multi-cores. GLSVLSI (2010).
[22] L. Mo and A. Kritikakou. 2019. Mapping imprecise computation tasks on cyber-physical systems. P2P Networking and Applications 12, 6 (2019).
[23] J. Pallister, S.J. Hollis, and J. Bennett. 2015. Identifying compiler options to minimize energy consumption for embedded platforms. Comput. J. 58, 1 (2015).
[24] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang. 2013. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. Trans. Comput.-Aided Des. Integr. Circuits Syst 32, 5 (2013), 695–708.
[25] B. Rouxel, S. Skalistis, S. Derrien, and I. Puaut. 2019. Hiding communication delays in contention-free execution for SPM-based multi-core architectures. In 31st ECRTS19.
[26] S.Z. Sheikh and M.A. Pasha. 2018. Energy-Efficient Multicore Scheduling for Hard Real-Time Systems: A Survey. TECS 17, 6 (2018).
[27] S.Z. Sheikh and M.A. Pasha. 2019. Energy-efficient multicore scheduling for hard real-time systems: A survey. TECS 17, 6 (2019).
[28] M. Thammawichai and E.C. Kerrigan. 2018. Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors. Real-Time Syst. 54, 1 (2018).
[29] U. Ullah Tariq, H. Ali, L. Liu, J. Panneerselvam, and X. Zhai. 2019. Energy-efficient Static Task Scheduling on VFI based NoC-HMPSoCs for Intelligent Edge Devices in Cyber-Physical Systems. TIST 1, 1 (2019).
[30] E. Vasilakis, I. Sourdis, V. Papaefstathiou, A. Psathakis, and M.G.H. Katevenis. 2017. Modeling energy-performance tradeoffs in ARM big.LITTLE architectures. 27th PATMOS (2017).
[31] H.E. Zahaf, A.E.H. Benyamina, R. Olejnik, and G. Lipari. 2017. Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms. J. of Systems Architecture 74 (2017).
[32] S. Zhuravlev, J.C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. 2013. Survey of energy-cognizant scheduling techniques. TPDS 24, 7 (2013).