Adaptation for Energy Saving in Time-triggered Systems Using Meta-scheduling with Sample Points *

Pascal Muoka†[0000−0003−1875−5017], Oghenemaro Umuomo†, Daniel Onwuchekwa†, and Roman Obermaisser†

†Institute for Embedded Systems, University of Siegen, Siegen, Germany

Abstract. Time-triggered systems offer significant advantages in embedded applications due to temporal predictability, implicit synchronisation and avoidance of resource contention. However, runtime adaptation of system services in these systems is motivated by energy efficiency without detriment to system performance. Too frequent adaptation result in more communication overhead. This work introduces a meta-scheduling technique with sample points to compute adapted static schedules for energy saving in time-triggered systems. An offline meta-scheduler optimises static schedules by applying slack events for energy saving at global periodic points in a schedule. The meta-scheduler maps the adaptation points to the runtime sampling period of adaptation. Slack events are reported synchronously by adaptation units at runtime, and adaptation is achieved through the aligned switching of component schedules facilitated by a Fault-Tolerant Agreement Protocol (FTAP). The meta-scheduler computes an MSG which holds all adapted schedules and describes the runtime switching of schedules based on reported slack events. The increased overhead due to periodic adaptation of the system schedule and energy saving of the meta-scheduling technique are evaluated. Results show a reduction in energy consumption compared with a base schedule while highlighting the trade-off between increased communication overhead and energy-saving.

Keywords: Energy saving · Communication overhead · Meta-scheduling · Adaptation · Time-triggered systems · Agreement protocol.

1 Introduction

Embedded system designers are increasingly faced with the demand for performance and dependability of embedded architectures in industrial automated systems to fulfil real-time requirements using less energy. Such energy saving must be achieved without detriment to system performance and dependability. Due to their determinism, embedded time-triggered (TT) architectures are beneficial for safety-critical applications and require runtime adaption for energy

* This work has been supported by the research project FRACTAL in part by the EC under grant number 877056 and the BMBF under grant number 16MEE015K.

GEFÖRDERT VOM Bundesministerium für Bildung und Forschung

savings [1–3]. Energy expended due to the execution of application tasks by processing elements (PES) in the system dominates the energy consumption of TT systems. Existing energy-saving approaches use power/clock gating, which degrades system performance [4]. To maintain real-time constraints, performance requirements, and energy savings, adapted system schedules and power/clock gating are used.

TT systems use static time-triggered schedules to execute tasks and route communications while avoiding timing conflicts. TT static schedules are computed at development time for temporal predictability and resource contention avoidance without dynamic arbitration. However, these systems save energy by switching active system schedules to adapted static schedules [1]. Static schedules ensure that all application tasks meet their deadlines. Furthermore, overestimating tasks' worst-case execution time (WCET) leads to MPSoC under-utilisation at runtime, creating gain times [5]. A task's gain time is the difference between its completion and WCET. In this work, gain times, referred to as slack events, are excess runtime computational periods of PEs guaranteed in a system schedule for executing application tasks.

Existing methods for utilising slack events use task remapping, or meta-scheduling [1, 6, 7]. A meta-scheduler computes adapted static schedules for runtime slack events. Adapted schedules are organised in a multi-schedule graph (MSG) and saved in the TT system's memory during runtime. The MSG is used at runtime to switch the active system schedule when slack events are reported and agreed on [8]. Unfortunately, this technique suffers from a state space explosion problem, exponentially increasing the number of computed adapted schedules and the memory needed to store the MSG at runtime.

Nevertheless, the adapted schedules and process for active schedule switch must preserve system temporal predictability, implicit synchronisation, and resource contention avoidance to achieve adaptation without introducing system failures. For consistent and aligned runtime adaptation, active system schedule switching is performed deterministically through an interactive consistency protocol (ICP) [9], instantiated synchronously by MPSoC adaptation units to agree on reported slack events. The ICP is frequently instantiated to minimise slack reporting delay. Although it minimises this delay, increased system sampling increases the number of adaptation messages exchanged between the adaptation units for a given system schedule.

This work examines the increased communication overhead caused by too frequent runtime invocations of a fault-tolerant agreement protocol (FTAP) in a double-ring topology. We propose a meta-scheduler that maps the FTAP adaptation points to a base schedule. These points, referred to as sample points, are used by the meta-scheduler to compute adapted schedules for all reported slack events. The proposed technique identifies an optimal sample period for each system schedule as a trade-off between energy savings and communication overhead.

This paper continues as follows. Section 2 discusses related works. Next, Section 3 describes the proposed approach to runtime adaptation, the agreement

protocol to establish a global system state, and a GA-based energy-saving meta-scheduler. Section 4 describes the experimental setup, and results are discussed in Section 5. Finally, Section 6 summarises the main contributions and findings.

2 Related Work

Sorkhpour et al. and Lenz et al. [6, 8] proposed an offline meta-scheduler for adaptive time-triggered systems. Meta-scheduler takes an application, platform, and context model and computes a multi-schedule graph (MSG). The MSG determines the next system schedule given the reported runtime slack event. In both works, the MSG size was a concern due to the storage constraints of time-triggered systems. However, their meta-scheduler did not consider an optimal sample period for adaptation while computing adapted schedules to maximise energy-saving and minimise communication overhead.

Lenz et al. [9] proposed an interactive consistency protocol for global runtime adaption. Local context events are synchronously broadcasted to adaption units in a ring-topology. All adaption units require an agreed system context for an aligned schedule switch. In their work, the increased communication overhead due to multiple invocations of the agreement protocol was not considered.

Fafoutis et al. [10] used static adaptive scheduling in Time Slotted Channel Hopping (TSCH), a MAC protocol that allocates time slots in static schedules to nodes. Under-allocation of time slots causes packet loss and poor energy efficiency, while over-allocation improves performance but wastes energy. Over-allocation and runtime adaptation save energy and support traffic bursts. This approach does not consider the communication overhead of the agreement protocol. Paterna et al. [11] proposed runtime adaptive task allocation for ageing systems. Their technique balances performance and energy usage to fulfil deadlines. A closed-form linear programming approach combined with approximate offline solutions is used online to extend the target MPSoC's lifetime.

Our work analyses the increased communication overhead resulting from frequent FTAP invocations for fine to coarse granularity. We combine the FTAP timing in meta-scheduling and compute an offline MSG for energy savings. This technique trades energy savings for runtime communication overhead.

3 Proposed Approach

An aligned system schedule switch is used to achieve energy-saving in time-triggered embedded systems without detriment to performance. Dynamic slack is utilised to start future tasks earlier, where runtime sampling of the system status results in reported slack events. Adapted system schedules utilising dynamic slack for energy-saving are computed offline to guarantee real-time requirements while avoiding timing conflicts in computing and switching schedules.

3.1 Adaptation in Time-triggered Multi-core Architectures

The time-triggered (TT) multi-core system architecture includes a network-on-chip (NoC) to facilitate message communication between PEs. The NoC's 2D-mesh topology connects PEs with bidirectional router links. PEs are instantiated as application cores in a homogenous architecture for executing tasks. Message-based ports offer an interface between the PEs and other components of the architecture, as illustrated in Fig. 1.

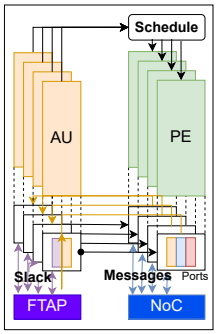


Fig. 1: Time-triggered Multi-core Architecture with Adaptation Unit

A static schedule prevents resource contention and maintains system synchronization through a global time base. Offline scheduling ensures tasks are completed before deadlines and messages through the NoC do not collide. The schedule allocates jobs and messages to system resources temporally, spatially, and contextually. Each task in the schedule is allocated to a PE and has a start time, WCET, and deadline. In addition, messages have injection times and an NoC routing path from source to destination.

Each PE in the TT system is connected to an adaptation unit (AU), which monitors tasks scheduled for the PE, as shown in Fig. 1. For tasks completed earlier than scheduled, a slack event is reported to the AU, which is the time difference between a task's worst-case execution time (WCET) and its execution time (ET) given by Equation 1.

$$slack = WCET - ET. \tag{1}$$

A crash event is reported when a PE can not execute a task. This work exploits only slack events for energy efficiency. When a slack event is reported, the active schedule is dynamically switched to a static energy-efficient schedule that uses the slack time to execute future tasks earlier than previously scheduled, generating idle time at the end of the schedule. Idle times are used to reduce energy usage by power/clock-gating the system. Due to the difficulties in estimating task execution time, WCETs are allocated pessimistically at design time resulting in runtime slack events. An offline meta-scheduler computes a

multi-schedule graph (MSG) for runtime slack events. Each system application cycle begins at the MSG root node, where no event is reported, and the system schedule is reset to the base schedule. MSG governs runtime schedule switching by considering the active node and the edge (slack event) to the next node.

The AU has a context monitor, a context agreement unit (CAU) and a schedule dispatcher. In addition, the AU is TT and scheduled independently to monitor and adapt the system schedule. The context monitor reports slack events to the CAU at scheduled times by encoding key identifiers of the event in a context bit string $lcontext_l$ illustrated in Fig. 2.

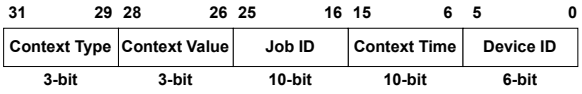


Fig. 2: Context Bit-String

In Fig. 2, *Device ID* and *Job ID* are the IDs of the local PE and scheduled task that originated the slack event. *Context time* is the system scheduling instant when the context monitor observes slack. *Context Value* represents the slack time compared to the tasks WCET given by Equation 1, and textitContext Type denotes a slack event.

All AUs are aligned for all schedule switches to prevent resource contention, message collision in the TT system, and system failure resulting from a loss of synchronisation. In addition, all AUs are aligned using a fault-tolerant agreement protocol (FTAP), where the reported slack event $lcontext_l$ is collected from the context monitor and sent to all CAUs for agreement. ICP is extended to FTAP to prevent NoC bandwidth overloading and to improve its fault tolerance, where all AUs are connected in a double-ring topology and reported slack events are broadcast and received in both directions. The schedule dispatcher holds the MSG and determines the following system schedule based on the active schedule and agreed slack event *gcontext*. The active system schedule and observed slack event are used as a mask to extract the following system schedule, which is dispatched based on the AU schedule.

3.2 Context Agreement

CAUs synchronously instantiate FTAP to avoid schedule switch-related system failure. FTAP broadcasts all reported slack events to all CAUs through their dedicated ports, as illustrated in Fig. 1. Active schedule switching requires a consistent view of the system state represented by *gcontext*. FTAP ensures all CAUs have the same system state view, enabling the schedule switch.

FTAP duplicates reported slack events $lcontext_l$ into $lcontext_{l1}$, $lcontext_{l2}$ and broadcasts them to neighboring CAUs through the double-ring network. When $lcontext_{l1}$ and $lcontext_{l2}$ are relayed to the origin CAU, FTAP converges.

FTAP compares $lcontext_{l1}$ to $lcontext_{l2}$ ensuring each duplicate bit is identical, converging them into $gcontext$ such that:

$$gcontext = \begin{cases} lcontext_l; & lcontext_{l1} = lcontext_{l2} \\ 0; & otherwise \end{cases} \tag{2}$$

Alg. 1 shows the FTAP pseudocode, where $lcontext_r$ represents a slack event from a CAU. $lcontext_{l1}$ and $lcontext_{l2}$ from the double-ring network are abstracted to $lcontext_l$. Each CAU relays all received slack events $lcontext_r$ until $lcontext_r = lcontext_l$. Each broadcast of a context bit-string $lcontext_r$, represented as a message msg_x from one CAU to the next, is recorded as a hop H_b. The number of messages $Tmsg$ broadcasted per instance of FTAP is given by Equation 3.

Algorithm 1: Fault-tolerant Agreement Protocol

```
   Input: lcontext_l
   Output: gContext
 1 if scheduled time then
 2 |  lContext = Input from context monitor;
 3 |  gContext ⊕ lcontext_l;
 4 |  broadcast lcontext_l to next neighbour;
 5 |  lcontext_r = lcontext_l from neighbour;
 6 |  while lcontext_r ≠ lcontext_l do
 7 |  |  gContext ⊕ lcontext_r;
 8 |  |  broadcast lcontext_r to next neighbour;
 9 |  |  lcontext_r = lcontext_r from neighbour;
10 |  end while
11 else
12 |  idle;
13 end if
```

$$Tmsg = H_b * N_{au}, \tag{3}$$

where N_{au} is the number of AUs in the TT system. $Tmsg$ represents the number of context bit-strings exchanged by the CAUs in each instance of the FTAP. FTAP is scheduled periodically to minimise the delay between reporting slack events and the schedule switch. Multiple instances of FTAP increase the number of context bit-strings exchanged by the CAUs given by Equation 4. C_{ov} is the total number of context bit-strings exchanged by the CAUs and represents the communication overhead of FTAP for a given system schedule period and FTAP frequency. A scheduled period is the time between the start of a schedule and the completion of the last task. $nSampPts$ represents the number of instances of FTAP for a given system schedule.

$$C_{ov} = Tmsg * nSampPts. \tag{4}$$

3.3 Meta-scheduling with Sample Points for Adaptation

An offline meta-scheduler computes energy-efficient schedules at periodic points for reported runtime slack events. These sample points map online FTAP timing to offline meta-scheduling. Globally applied sample points ensure consistent schedule adaption and mapping of FTAP timing to meta-scheduling.

Meta-scheduler guarantees temporal constraints by fixing the parent schedule scheduling decisions before each sample point. This approach prevents resource contention while computing an energy-efficient schedule where future tasks are executed earlier. The meta-scheduler computes schedules S_i utilising an application, platform, and context models (CM). Each computed schedule is a node in the multi-schedule graph (MSG). An edge in the MSG is a slack event for which an optimised child node is computed from the parent node.

The AM is a model of the runtime application represented as a tuple $< T, M >$. Each vertex $j_i \in T$ is an application task. $m_{i,k} \in M$ is a task-to-task message, as shown in Fig. 3. $< WCET, D >$ represents a task's worst-case execution time and deadline. A message $m_{i,k}$ is a tuple $< S, R, M_{sz} >$ where S and R are the sender and recipient tasks and M_{sz} is the message size.

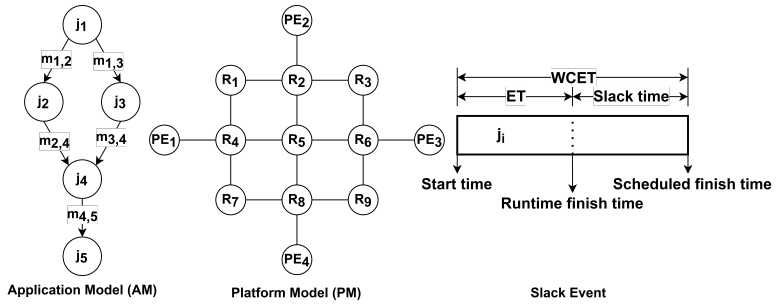


Fig. 3: Application Model, Platform Model and Slack Event

The PM represents the TT multi-core system and is modelled as an undirected graph G_p represented as a tuple $< N, L >$. A vertex is either a PE $PE_i \in C$ for the execution of tasks or a router $R_i \in V$ to facilitate the exchange of messages between the PEs such that $N = C \cup V$. An edge $l \in L$ represents the bidirectional physical link of the TT multi-core system interconnecting all routers and PEs. The set of routers and their physical interconnect constitutes the NoC which has a 2D-mesh 3X3 topology.

The CM is a list of runtime slack events represented as a tuple $< j_i, sl_t >$ where sl_t is the slack time of task j_i expressed by Equation 1. The meta-scheduler generates a multi-schedule graph (MSG) using the AM, PM and CM. The MSG is a directed graph of schedules which are the nodes of the graph, and slack events describe the edges of the MSG. Schedules are dispatched during runtime based on the active node in the MSG and the reported slack event.

The proposed meta-scheduler uses a genetic algorithm (GA) to compute a new schedule. The GA's objective function is to minimise the schedule makespan and maximise energy saving where idle times are used for power/clock gating. The GA's objective function ensures all computed adapted schedule maintains performance requirements while maximising energy saving. Initially, a base schedule S_0 is computed for a given AM and PM when no event is reported, and scheduling decisions are not decided. Then, the base schedule S_0 is added to the MSG as the root node, and a calendar of events Cal is created from the CM and S_0 as described in Alg. 2. The calendar of events Cal details the reported times of all slack events $sl_t \in$ CM.

Algorithm 2: Meta-scheduling Procedure

Input: AM, PM, CM
Output: MSG
1 initialise MSG;
2 $S_0 = GA(AM, PM, V_d)$;
3 add S_0 to MSG;
4 Cal = create calendar (CM, S_0);
5 **Call Algorithm 3**;
6 **return** MSG

At each sample point, the meta-scheduler checks for the occurrence of slack events and replace their occurrence time with the sample point time. The difference between slack occurrence time and the sample point time, δ, represents a delay in slack reporting, leading to a reduced slack time. The meta-scheduler ignores a slack event observed after the $WCET$ of its task and is not reported as it is no longer beneficial for adaptation. The δ and number of ignored slack events are minimised by increasing the frequency of FTAP, leading to increased overhead.

Scheduled WCET of jobs in the AM $WCET$ are replaced with the sample point time based on the reported slack events. Then, as described in Alg. 3, a new schedule is computed from the updated AM $AM^{'}$. The active schedule S_i is used to fix decision variables V_d before the sample point in the new scheduling problem. The decision variables consist of task allocation to PEs, task start times, message routing through the NoC, and message injection times. Finally, Alg. 3 is recursively called with the updated models and calendar, the new schedule and MSG.

4 Experimental Setup

A time-triggered multi-core node is set up to observe the communication overhead of FTAP for different timing granularities. Then, these timing granularities are mapped to the meta-scheduler to compute adapted energy-saving schedules.

Algorithm 3: GA-based Meta-scheduler

Input: AM, PM, Cal, S_i, MSG
Output: MSG
if *Cal is empty* **then**
  return
else
  $sl_t, sl_u, \ldots, sl_z$ = all events in Cal before next sample point;
  Cal' = remove $sl_t, sl_u, \ldots, sl_z$ from Cal;
  Call Algorithm 3(AM, PM, Cal', S_i, MSG);
  AM' = apply $sl_t, sl_u, \ldots, sl_z$ to AM;
  fix decision variables $V_d \; \forall \; j_i, m_{ik}$ before each sample point;
  $S_i = GA(AM', PM, V_d)$;
  add S_i as node and $sl_i, sl_k, \ldots, sl_n$ as edge to MSG;
  Cal'' = update $Cal'(S_i)$;
  Call Algorithm 3(AM', PM, Cal'', S_i, MSG);
end if

4.1 Time-triggered Multi-core Architecture

The MPSoC architecture illustrated in Fig. 1 was instantiated using VHDL in the programmable logic of the Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit. Four ARM-based processing cores were interconnected through a TTNoC [12]. Adaptation units were instantiated using the Vivado toolchain for each core in the MPSoC.

To evaluate the performance of FTAP, we simulated a schedule of 20 and 100 tasks with hard deadlines on the TT MPSoC architecture. Various timing granularity of FTAP in the range $100\mu s$ to $500\mu s$ was applied, and the communication overhead due to the exchange of slack events was observed. $100\mu s$ represents a more frequent execution of the FTAP than $500\mu s$. The communication overhead criterion was chosen as it reflects the effectiveness of FTAP as a broadcast protocol. For each simulation of a schedule on the TT MPSoC, the total communication overhead was evaluated as the number of context bit strings $lcontext_r$ exchanged between the adaptation units. Assuming constant energy dissipated in the exchange of each context bit string, energy saving is achieved through an offset between energy saving from meta-scheduling and the total communication overhead due to instances of FTAP.

The total communication overhead C_{ov} resulting from multiple instances of FTAP is given by Equation 4. $nSampPts$ represents the number of instances of FTAP for a given schedule.

4.2 Meta-scheduler with Sample Points

Tasks are scheduled based on their WCET to ensure sufficient computational slots for completing tasks. This pessimism in scheduling tasks results, on average, in 70% of application tasks completed in 50% of their WCET [13]. The difference

between a tasks completion time and its scheduled WCET gives its slack time given by Equation 1. This slack time constitutes a slack event of the task and is described in the context model (CM). MSGs are therefore computed for slack events of 50% slack time due to the difficulty in offline estimation of runtime task execution times. The size of the CM is also limited to 10 slack events to evaluate the meta-schedulers energy saving.

The meta-scheduler was implemented in C++ using a GA-based scheduler [14, 15]. The GA is set to a population size of 3000 and a number of generations of 200, resulting in better scheduling solutions. In addition, probabilities of crossover and mutation are set to 0.4 and 0.01, respectively, preventing a random search of the GA.

Experiments were conducted using the OMNI computing cluster of the University of Siegen [16] for an AM consisting of 20 and 100 tasks and messages, with average WCETs between $700\mu s$ and $1000\mu s$. The PM is implemented as a 4-core 3x3 NoC mesh platform connected in a cross topology. The CM consists of 10 reported slack events distributed over tasks in the AM. The AM and PM are generated using the SNAP library [17].

To evaluate the impact of timing granularity of FTAP, we computed MSGs with sample points with granularity in the range $100\mu s$ to $500\mu s$. The chosen range ensures that all tasks in every schedule are sampled for a slack event at least once in a scheduled period, and sufficient time is available to adapt the schedule based on the reported slack event.

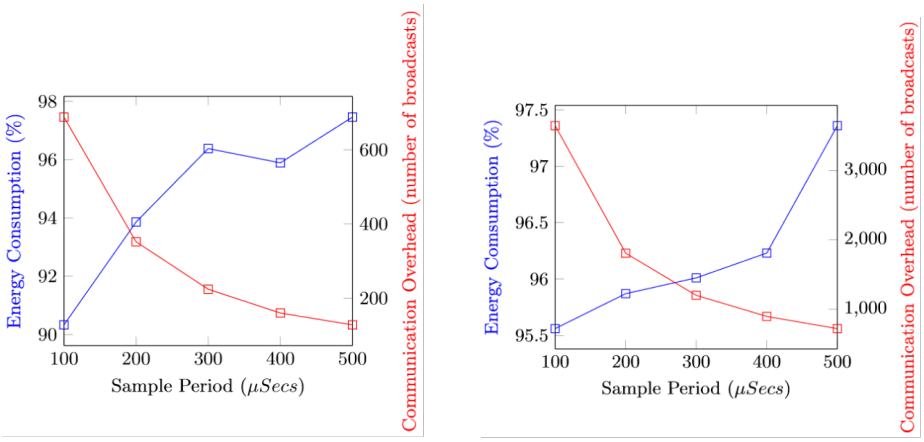
Energy efficiency is estimated as the percentage difference in makespan between the base schedule in a computed MSG and the last schedule (idle time), representing a path in the MSG from the root node to the leaf node. Idle times are utilised at runtime to power/clock gate the system, saving energy that would have been expanded in executing the application.

5 Results and Discussion

In Figs. 4a and 4b, we show the energy consumption of adapted schedules of the meta-scheduler. For each leaf schedule in the MSG, the makespan is compared to the makespan of the base schedule. For example, a 100% energy consumption indicates no reduction in the leaf schedule makespan, whereas a 50% energy consumption is characterised by a leaf schedule makespan half of the base schedule makespan. The energy consumption of the leaf schedule is evaluated for the timing granularity of FTAP in the range $100\mu s$ to $500\mu s$ for AMs with 20 and 100 tasks. The difference in the energy consumption of the adapted schedule to the base schedule represents the energy saved by switching to the adapted schedule.

The energy expanded in executing application tasks dominates the energy consumption of the system compared with the energy expanded in exchanging messages between agreement units. Therefore, the system's energy consumption reduction is observed when more slack events are adapted. An increase in FTAP frequency leads to overutilisation of the adaptation network bandwidth, thus increasing communication overhead. The communication overhead increases with

a decrease in the sample period. Similarly, the meta-scheduling algorithm with sample points resulted in higher energy saving of the adapted schedules with a sample period of finer granularity. A key point in Fig. 4a, $\approx 180\mu s$, indicates a balance between the energy consumption and communication overhead. A decrease in the sample period from this point, although resulting in higher energy consumption, increases the communication overhead of FTAP. Conversely, an increase from this point results in lower energy saving and communication overhead. This result is also observed in Fig. 4b, where the key point is $\approx 270\mu s$.



(a) 20 Tasks (b) 100 Tasks

Fig. 4: Energy Saving and Agreement Overhead

An increase in the density of reported slack events, ρ results in increased energy saving observed by comparing Fig. 4a with Fig. 4b. In Fig. 4a, 10 slack events reported for 20 tasks, $\rho = 50\%$, results in $\approx 10\%$ energy saving compared with Fig. 4b where $\rho = 10\%$ results in $\approx 5\%$ energy saving. For each combination of the AM, PM, and CM, an optimal sample period can be selected to maximise energy saving while minimising the communication overhead of FTAP.

6 Conclusion

This work investigated the energy saving of time-triggered adapted schedules due to the periodic execution of a Fault-tolerant Agreement Protocol (FTAP) in an adaptive time-triggered multi-core architecture. Constraints such as communication overhead was considered in determining an optimal sample period for a given application. The periodic runtime execution of adaptation is mapped to a meta-scheduler's schedule generation. Schedules are adapted for energy saving, and the number of static schedules computed for runtime adaptation is observed.

The meta-scheduler computes adapted schedules for slack events reported before each sample point as a result of superimposing runtime timing of FTAP on meta-scheduling. An optimal sample period for a given application scenario is determined as a trade-off between energy saving, communication overhead and the number of schedules needed for adaptation.

References

1. Obermaisser, R., Ahmadian, H., Maleki, A., Bebawy, Y., Lenz, A., Sorkhpour, B.: Adaptive time-triggered multi-core architecture. Designs **3**, (2019)
2. Heilmann, F., Syed, A., Fohler, G.: Mode-changes in COTS time-triggered network hardware without online reconfiguration. SIGBED Rev. **13**, 55–60 (2016)
3. Fohler, G., Gala, G., Pérez, P.-G., Pagetti, C.: Evaluation of DREAMS resource management solutions on a mixed-critical demonstrator. In: 9th European Congress on Embedded Real Time Software and Systems (ERTS). (2018)
4. Gaillardon, P., Beigne, E., Lesecq, S., Micheli, G.: A survey on low-power techniques with emerging technologies: From devices to systems. In: ACM J. Emerg. Technol.Comput. Syst. **12.2**(12). (2015)
5. Audsley, N.-C., Davis, R.-I., Burns, A.: Mechanisms for enhancing the flexibility and utility of hard real-time systems. In: 5th IEEE RTSS, pp. 12–21. (1994)
6. Sorkhpour, B., Murshed, A., Obermaisser, R.: Meta-scheduling techniques for energy-efficient robust and adaptive time-triggered systems. In: IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), pp. 0143–0150. (2017)
7. Zou, Y., Pasricha, S.: HEFT: A hybrid system-level framework for enabling energy-efficient fault-tolerance in NoC based MPSoCs. In: International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 1–10. (2014)
8. Lenz, A., Pieper, T., Obermaisser, R.: Global adaptation for energy efficiency in multicore architectures. In: 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp. 551–558. (2017)
9. Lenz, A., Obermaisser, R.: Global adaptation controlled by an interactive consistency protocol. Journal of Low Power Electronics and Applications **7**, (2017)
10. Fafoutis, X., Elsts, A., Oikonomou, G., Piechocki, R., Craddock, I.: Adaptive static scheduling in IEEE 802.15.4 TSCH networks. In: IEEE 4th World Forum on Internet of Things (WF-IoT), pp. 263–268. (2018)
11. Paterna, F., Acquaviva, A., Benini, L.: Aging-aware energy-efficient workload allocation for mobile multimedia platforms. In: IEEE Transactions on Parallel and Distributed Systems **24**(8), pp. 1489–1499. (2013)
12. Ahmadian, H., Obermaisser, R., Abuteir, M.: Time-triggered and rate-constrained on-chip communication in mixed-criticality systems. In: Proceedings of the 10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 117—124. (2016)
13. Axer, P., Ernst, R., et al.: Building timing predictable embedded systems. In: ACM Transactions on Embedded Computing Systems (TECS) **13**(4). (2014)
14. Wall, M., Galib, A.: A C++ library of genetic algorithm components. Boston: Mechanical Engineering Department Massachusetts Institute of Technology. (1996)
15. Muoka, P., Onwuchekwa, D., Obermaisser, R.: Adaptive scheduling for time-triggered network-on-chip-based multi-core architecture using genetic algorithm. Electronics **11**, (2022)

16. Universität Siegen, https://cluster.uni-siegen.de/omni/. Last accessed 2 July 2022
17. Leskovec, J., Sosič, R.: SNAP: A general-purpose network analysis and graph-mining library. In: ACM Trans. Intell. Syst. Technol. **8**. (2016)