# Scenario-Based Meta-Scheduling for Power Performance Optimization Supporting Core and Router Frequency Scaling in Time-Triggered Multi-Core Architectures

Babak Sorkhpour, *Student Member, IEEE*; Roman Obermaisser, Raimund Kirner, *Senior Member, IEEE*

***Abstract***— **Complex electronic systems are used in safety-critical applications (e.g., aerospace, nuclear stations), for which the certification standards demand the use of assured design methods and tools. Meta-scheduling is a way to manage the complexity of adaptive systems via predictable behavioural patterns established by static scheduling algorithms. This paper proposes a meta-scheduling algorithm for adaptive time-triggered systems based on Networks-on-a-Chip (NoCs). The meta-scheduling algorithm computes an individual schedule for each dynamic event of slack occurrence. Each dynamic slack occurrence triggers the shift to a more energy-efficient schedule. Dynamic frequency scaling of cores and routers is used to improve the energy efficiency, while preserving the temporal correctness of time-triggered computation and communication activities (e.g., collision avoidance, timeliness). Mixed-Integer Quadratic Programming (MIQP) is used to optimise the schedules Experimental results for an example scenario demonstrate that the presented meta-scheduling algorithm provides on average a power reduction of 34%. Our approach was able to deploy 93 dynamic slack schedules compared to the single slack schedule of using static slack scheduling.**

***Index Terms*— MPSoC, NoC, MIQP, scenario-based, meta-scheduling, mixed-criticality, time-triggered.**

## I. INTRODUCTION

Embedded systems are used in a wide range of industrial systems, for example, aerospace, medical and automotive systems. Many premium carmakers plan to invest heavily in e-cars, which significantly depend on embedded systems [1]. However, with the new IoT era, minimizing power consumption becomes a primary concern of system designers. Scheduling optimisation is used to help engineers and system designers to increase the energy-efficiency and improve the behaviour of a system [2]. Scenario-based scheduling is a scheduling technique [3] to predict, calculate and model the circumstance events in safe-critical systems. Multi-Processor System-on-Chip (MPSoC) systems typically represent one of the most power consuming components of embedded systems, and most researchers focused on reducing power and energy consumptions of computational cores. Today, MPSoCs typically support the scaling of frequency and voltage (e.g., DVS, DVFS) as well as multiple sleep states for cores. However, frequency tuning for both cores and routers on multi-core architectures (e.g., NoC) so far has been an open research challenge.

In this research, we extend scenario-based scheduling on MPSoC with time-triggered communication to provide optimisation of energy efficiency not only for cores but also for NoC routers. The energy efficiency optimisation uses dynamic frequency scaling, where we are able to scale the frequency of each core and router individually. This algorithm is suitable for mixed-criticality [4] and safety-criticality [5] by supporting fault-tolerant applications and adaptive systems. Compared to purely static scheduling, our approach provides more energy efficiency and enhanced flexibility.

Compared to our own previous research [3, 6, 7] and [8], this work does not focus on introducing a new architecture for scheduling or meta-scheduling technique. Instead, we provide an improved, extended, more flexible and reliable meta-scheduling architecture [9, 10] for MPSoCs and NoCs.

The novelty of this paper is to extend energy efficiency optimisation of scenario-based meta-scheduling for MPSoCs with time-triggered communication by not only providing frequency scaling of cores, but also of NoC routers.

Compared to our previous work [3, 6, 7] and [8, 11], which was only able to assign a single task per core, the algorithm presented in this article is now also able to assign multiple tasks per core on a multi-core platform. As part of our solution, we had to develop a scenario-based graph traversal algorithm and tool that called MeS [3] and a backtracking algorithm based on this tool for identifying and managing events. We build a meta-scheduling tree, with the top node being assuming no occurrence of dynamic slack. Each occurrence of dynamic slack switches to another schedule down the tree with better energy-efficiency.

"The European H2020 project SAFEPOWER has supported this work under the Grant Agreement No. 687902."

The authors are with the Department of Electrical Engineering and Computer Science, University of Siegen, Germany (babak.sorkhpour@uni-siegen.de, roman.obermaisser@uni-siegen.de) and the Department of Computer Science, University of Hertfordshire, UK (r.kirner@herts.ac.uk)

The method and algorithm presented in this article is of direct benefit for Green IT [12–15]. With the goal of Green IT to save the environmental economy of resources, the power consumption is one of the most important parameters.

This article is organized as follows: The basic concepts and related work are described in Section II. Section III describes the meta-scheduling system model. Scheduling techniques and scheduling for energy-efficiency are explained in Section IV. The implementation of algorithms for meta-scheduling and its features are explained in Section V. Experimental results based on a case study are presented in Section VI. Section VII concludes the article.

## II. Basic Concepts And Related Work

*Multi-Processor System-on-Chip* (MPSoC) in heterogeneous systems include many elements, e.g., CPU, GPU, and *Network-on-Chip* (NoC). With latency-sensitive applications running on such MPSoC platforms, the cores and routers must be designed and operated accordingly to satisfy the performance requirements. *Dynamic Voltage and Frequency Scaling* (DVFS), adaptive routing and frequency scaling in routers and links can potentially improve energy efficiency and performance of NoC. We further notice that the execution time variations of tasks sometimes lead to dynamic slack time [16], which can be exploited by different levels of execution times that can be changed without causing a performance penalty. In this work, we take advantage of the slack time in cores to reduce energy consumption. Routers can work at a lower frequency without causing a performance penalty at the system level.

In this section, we review the state of the art of related works and briefly introduce our own previous work [6, 8, 17–20] . In addition, we present our meta-scheduler tool MeS and a new strategy and model to improve the energy-efficiency using both cores and routers in the schedules.

Many algorithms, methods, and techniques were proposed for scheduling distributed embedded real-time systems. Schedulability analysis is a primary part of real-time scheduling. In particular, time-triggered systems depend on static schedules that define the use of computational and communication resources based on a global time base. In [18, 21], Kopetz explains how the correctness of a time-triggered system depends also on the timing of the computational results. Time-triggered communication of messages ensures predictability and resource adequacy. Time-triggered control is a valuable solution in safety-critical systems to manage the complexity and provide analytical dependability and timing models.

A group of tasks and messages are said to be schedulable with a certain scheduling method, if enough resources (e.g., cores, routers) and slot time is available to execute all these tasks and transmit all messages before their deadlines. Each real-time task and message is assigned a priority and a deadline which are defined in an Application Model (AM). In the time-triggered paradigm [18] of real-time scheduling, processes are controlled and organized by the progression of time only, and a schedule is designed for the total duration of a system's execution. One of the typical techniques used for time-triggered systems is using schedule tables. They are easy to verify and thus favourable in safe-critical systems that must be certified [22].

However, adapting to significant events within the computer system or in the environment is a challenge in time-triggered systems. In [19], Fohler presents a method for supporting schedule changes based on operational modes by switching and traversing among static-schedules in a schedules status tree. Thereby, a pre-run-time and the scenario-based scheduled system can adapt to changes of the environment. Isakovic et al. explained that physical component designs and interfaces with a specific computer system should provide a clean design methodology approach, but systems get more complex primarily when working with heterogeneous systems and protocols [23]. However, it is challenging to ensure that the functional properties match the system specifications and regulatory guidelines. The authors also offer a mixed-criticality integration solution based on a time-triggered architecture on a hybrid system-on-a-chip [24] platform. However, their method does not provide a scheduling technique for energy-efficiency in heterogeneous systems.

*Power-efficiency for NoCs*: Sheikh et al., worked on the combined optimization of performance, energy, and temperature [25]. They proposed an optimization framework called PET. The authors find efficient solutions using a multi-objective evolutionary algorithm and a Strength Pareto Evolutionary Algorithm for task scheduling and voltage selection. They worked with multiple cores to do frequency switching. Jingcao Hu et al. used static schedules for both communication and computation of tasks on heterogeneous NoC architectures for multimedia. Their algorithm achieved about 44% energy on average, compared to the standard earliest-deadline-first scheduler [26].

*Task slack and procrastination*: Jejurikar et al. take the slack time [16] of tasks into account to improve the deadline satisfaction and to reduce the energy consumption [17]. They propose an algorithm for fault-tolerant resource allocation in real-time dynamic scenarios. On an average, they reduce the energy consumption by 29.1% and 6.7%, compared to previous work.

*Frequency tuning on NoC*: In the literature [27], *DVFS* is one of the most famous energy-efficiency techniques for improving the power-efficiency of chips at run-time [28]. We also use DVFS to establish optimized frequencies for cores and routers. Chai et al. worked on the combination of execution time, *DVFS*, slack time, and power consumption to find energy-efficient schedules with minimised processor frequencies [29]. Li et al., used a genetic algorithm (GA) to achieve near-optimal voltage and frequency assignments for considering the problem of energy-efficient contention-aware application mapping and scheduling on NoCs [30]. Their reported results show that jointly utilizing dynamic voltage scaling on processors and frequency tuning on NoC links provides excellent potential for overall energy reduction in MPSoCs and the overall system energy consumption is significantly reduced. Lee et al. worked on energy-efficient scheduling for *DVFS-enabled* multi-core architectures, which saved energy by executing the tasks in parallel and by down-scaling of the frequency [31]. When a task is executing on a single core, their approach reduces the energy consumption significantly: up to 67% by considering the non-linear scaling and the finitely discrete energy consumption rates of available frequencies. But their model does not support scenario-based scheduling and frequency scaling of communication routers.

*Power consumed by NoCs:* Chai et al. presented that on average 28%~36% of the total power consumption depends on NoCs [29]. Han et al. presented a low power methodology and routing algorithm regarding temperature to achieve an ultra-low-power NoCs [32]. Experimental results demonstrate an average power reduction of 36.0% over 21 applications. Hangsheng et al. show by experimental results that the communication interconnect can consume up to 36% percent of the power in an MPSoC [33]. Tariq et al. have investigated the problem of scheduling and energy-aware mapping via Integer Linear Programming (ILP) for a set of tasks with individual deadlines and conditional precedence constraints on a heterogeneous NoC-based MPSoC to achieve minimizing the total expected energy consumption of all the tasks [34]. The authors have used a polynomial-time heuristics to achieve improvements of 31% and 21% regarding energy reduction. Today's NoCs have become an essential building block of multi-core architectures. NoCs are composed of three main building blocks: links, network interfaces (NI) and routers. More cores typically means a higher power consumption. Including more cores thus presents a design hurdle, architectural complexity and higher energy consumption. For example, the KALRAY MPPA2-256 is made of up to 288 cores: 256 computing cores, 16 management cores, and four quad cores (see Figure 1). Kalray's MPPA [35] technology addresses these challenges by combining high-performance cores with low-power processors [36].

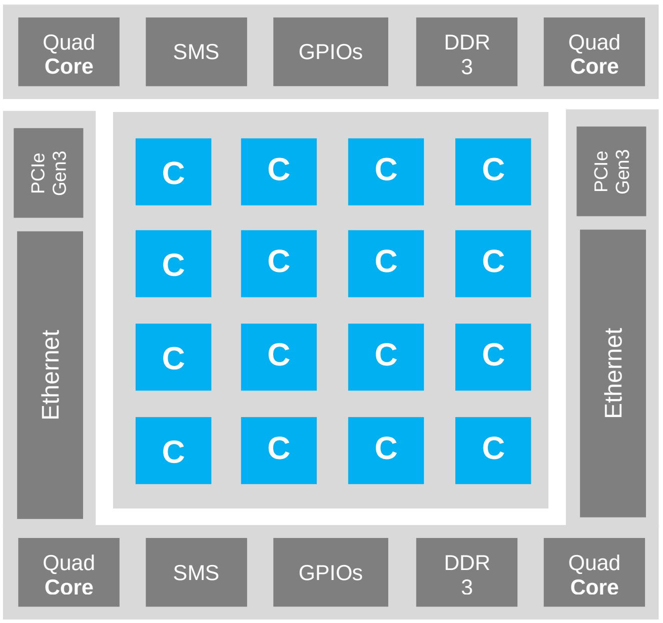


*Figure 1. Kalray's MPPA network-on-chip (The MPPA2®-256 Bostan2 processor [36]*

*Mixed-Integer Quadratic Programming (MIQP):* In [37] mentions the use of the *MIQP* for solving scheduling problems. With *MIQP* the objective function is quadratic with respect to the integer and continuous variables, while the constraints are linear with respect to the variables of both types.

*Static-scheduling formulation:* Our meta-scheduler presented in this article is a static-scheduling algorithm. Murshed et al. [8] have provided a static message-based scheduling approach that guarantees the absence of collisions in message routing for a single task per core. Static-scheduling is a reliable solution for robust and timely systems, especially in time-triggered systems. Their scheduling problem was established by formulating a *Mixed-Integer Linear Programming* (*MILP)* problem. In our previous work we have extended it to *MIQP* by calculating and reformulating the objective function, constants, decision variables, constraints [3, 7]. In [3, 8] the *IBM ILOG CPLEX* optimizer [38] has been used to solve the *MILP* respective *MIQP* problems. Since *CPLEX* itself can only solve linear problems, we defined the objective function using a linearization technique [39–41] to find the linear approximation. LP/IP/QP can provide optimal answers in minimization or maximization scheduling problems, and they can use in scheduling problems when constraints, variables, and objective functions are linear or quadratic. Like in our previous work, in this research we are using *MIQP* as our goal is to achieve optimum solutions for time-triggered and embedded systems. Optimum solutions can provide better solutions than feasible answers or other methods, like Genetic Algorithms (*GA)* etc. Majd et al. used *GAs* as they have believed it can better locate a near optimal than a list-schedule [42]. They reported that many existing approaches do not consider communication cost when applying *GA* to MPSoC scheduling problem. In contrast, we consider also the communication costs. To handle the communication delays between processors, Majd et al. used a combination of *GA* and the Imperialist Competitive Algorithm, while we are using *MIQP* together with the above mentioned linearization method.

*Meta-scheduling:* Meta-scheduling can be described as a technique to optimize the computational workload by combining and organizing multiple distributed resources in an integrated view. In other words, it is an extended data-flow model and quasi-static scheduling for dynamic behaviour changes. Most of the research on meta-scheduling has been done for enterprise grids, clouds, and data centers, for example, GridWay, community scheduler frameworks, Moab cluster suite, Maui cluster scheduler, DIOGENES, synfiniWay's meta-scheduler [43]. On the other hand, Jung et al. worked on meta-scheduling for green computing to reduce the energy consumption and thus reduce the $CO_2$ emission [44] into the atmosphere. They call their approach GreenMACC [45]. Jung et al. [46] used for their model the Synchronous Data Flow [47], which is commonly used in signal processing or streaming applications. Their model can be used in the dynamic behaviour changes and classified as multi-mode dataflow models. The proposed technique is used to minimize the number of required processors for multiprocessor scheduling by considering task migration between modes. The focus on minimisation of required resources, but not on parameterisation of resources, like frequency scaling.

To summarise, the existing work on meta-scheduling and quasi-static scheduling, low-power and energy-efficient scheduling focus on different models, methods, and architectures than ours. For example, with/without DVS, DVFS or dynamic power management capabilities and tries to dynamic or static manipulate the task execution slacks to exploit them. Since we aim to use scenario-based meta-scheduling for a wide range of applications, like energy-efficient and fault recovery, within an integrated tool, the existing approaches cannot be directly applied to our scenario-based meta-scheduling where they target architectures or modelling are entirely different.
For example, while our work is related to [26], we not only cover static slack time, but also support dynamic slack time and scenario-based scheduling. We are using slack time of tasks to calculate the best *slowdown factor* for the communication and computation to maximise energy efficiency. However, in [26] they use the slack time for different paths, and allocate the slack time to different tasks.

## III. META-SCHEDULING SYSTEM MODEL

In the following we briefly describe our meta-scheduling tool MeS [3]. MeS allows to optimise the placement of tasks and messages to optimise execution times and injection times. MeS has been improved by adding the new optimisation model as described in Section IV.

*Dependability requirements* which are covered by MeS contain detect, isolate or mitigate errors, transient physical problems (e.g., core fault), that occur in system or schedules.

*Timing requirements* which are covered by MeS contain time constraints, execution time determinism, predictability, composability, flexible worst-case response for a slack time by static analysis.

*Independent of network topology:* MeS scheduling and routing method is independent of the hardware platform and can cover the topology of different networks (e.g., mesh, direct network, indirect network, balanced tree).

*Input models for MeS*
*Scheduling Model (SM)*: The SM, as shown in Figure 2), contains specific data structures which are the physical model (PM), the application model (AM) and context model (CM).

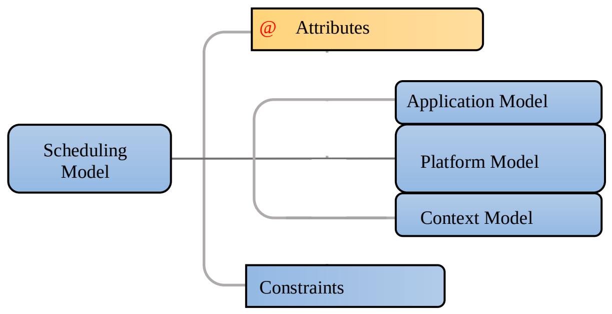


*Figure 2. Data structure of MeS schema model*

*Application Model (AM)*: The AM describes application dependencies at the software layer. It presents task and messages priorities, dependencies, and hierarchy, which the system designer is using to shape the AM section of the input file.

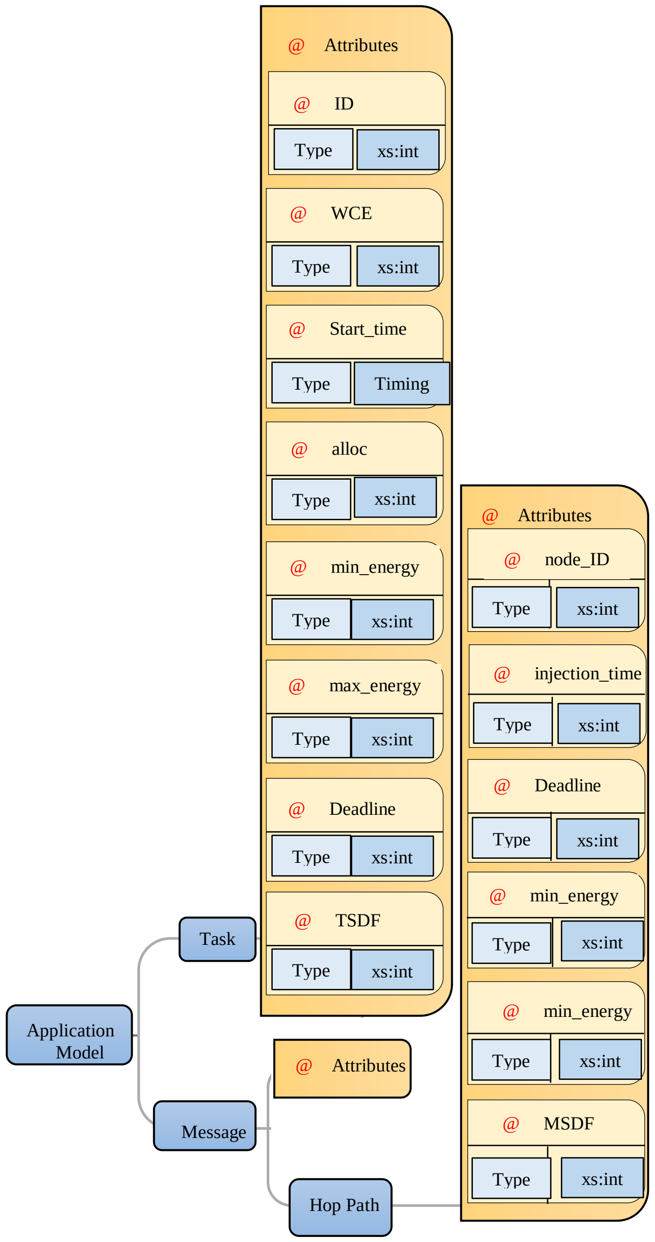


*Figure 3. General AM schema*

The AM has two main elements: tasks, and messages. Figure 3 presents that each element contains specific attributes with built-in derived type. e.g., task contains ID, WCET, start time, allocation, min and max energy, deadline, and slowdown factor; message contains message ID, sender and receiver ID, slowdown factor, hops which all data type in this work are an integer.
*Platform Model (PM)*: The PM describes the physical dependencies at the hardware layer. Figure 4 shows a platform example, describes nodes and links priorities, dependencies and hierarchy, which the system designer uses to shape the PM

section of the input file.

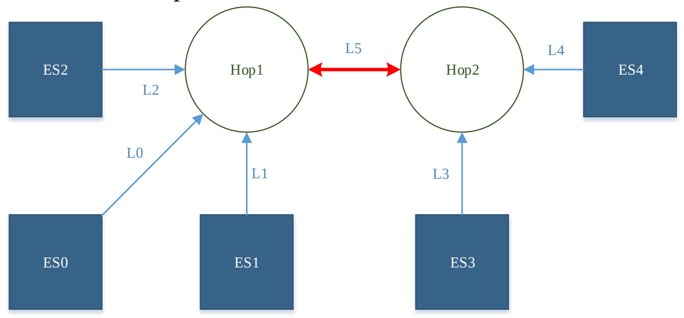


*Figure 4. Conceptual PM*

The PM has two main elements: nodes, and links. Figure 5 presents that each element contains specific attributes with built-in derived type; e.g., the node contains ID, type (router or core), frequency and link contains ID, from node and to node.

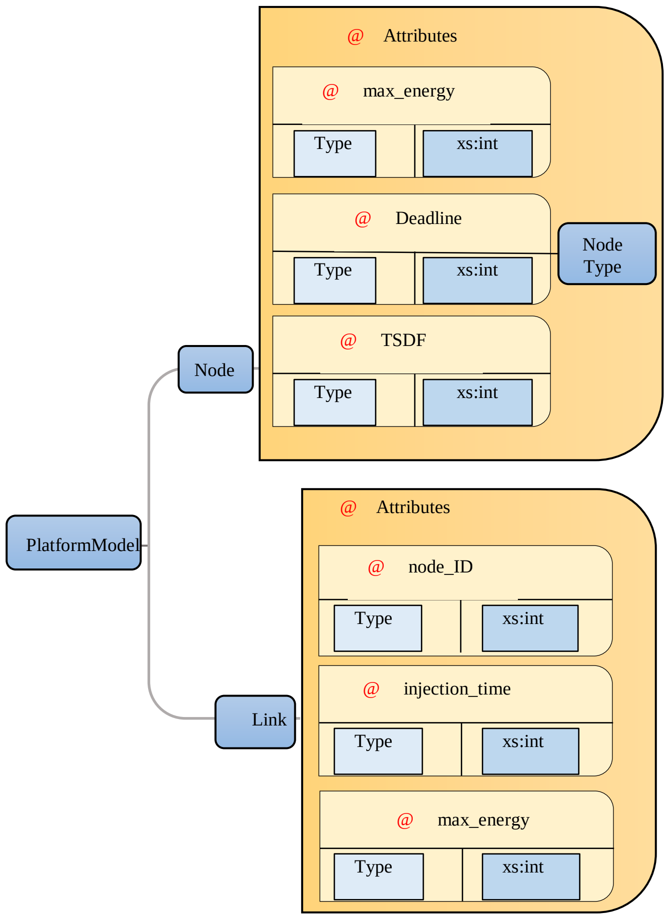


*Figure 5. General PM Schema*

Context Model *(CM)*: The CM is describes which scenarios in the system management layer are possible and can cover when single or multi faults and events happened. In details, the CM describes every faults and event for each element in the PM and the AM, including priorities, dependencies, timing, and hierarchy. The CM has three main elements: slack, energy, and fault. Figure 6 presents that each element contains specific attributes with built-in derived type. e.g., slack event contains the new execution time and related task ID, energy event contains energy level, and fault including node, link, and core fault which they are representing related object ID (e.g., link ID) which all data type in this work are an integer.

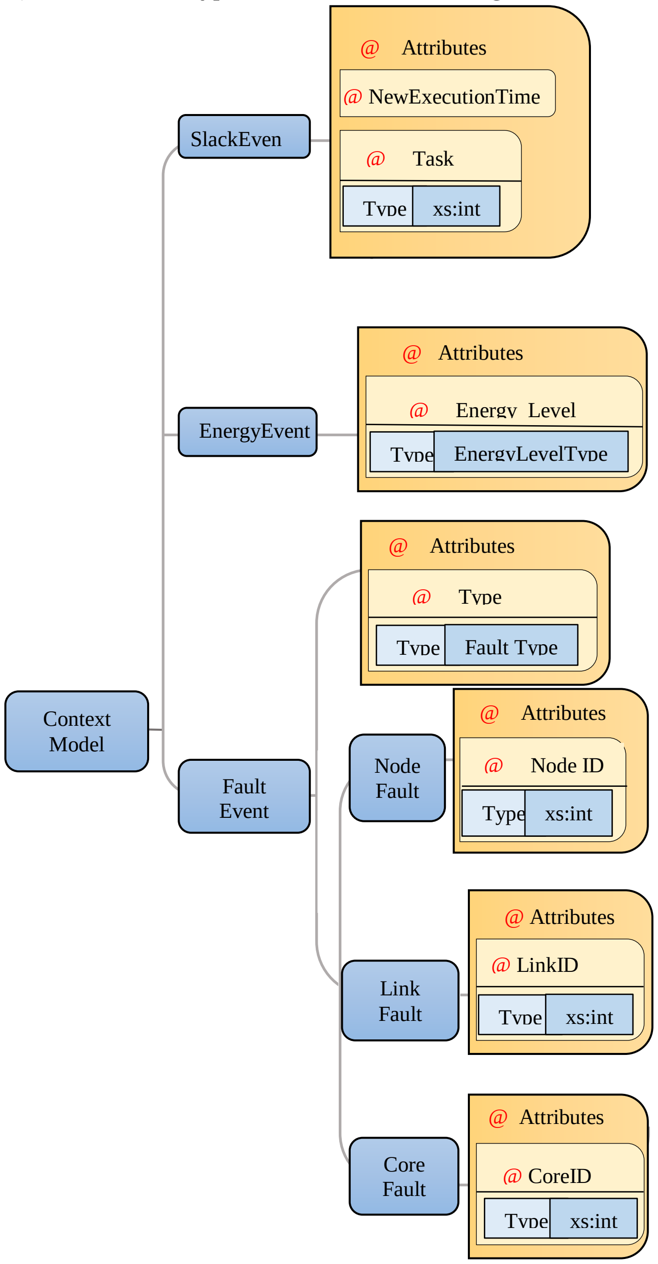


*Figure 6. General CM schema*

## IV. Scheduling for Energy-Efficiency

The provision of fault tolerance and energy-efficiency are essential for MPSoCs, like NoCs and grids, where a permanent failure, e.g., of a core or router, might occur during the execution of scheduled tasks and messages. The schedules can be generated statically before their execution with the help of a schedules-event graph, which represents the schedules and dependencies between them and events. In [48, 49] Eitschberger et al. presents a scheduling model to balance between faults and energy to maximize the performance in static schedules. However, it is critical to minimize the length of a schedule (the so-called *Makespan* [50]), i.e. the duration till all jobs have finished processing, while integrating fault

tolerance techniques. Fault tolerance techniques typically result in performance overhead, which furthermore leads to an increase of the Makespan [51].
One of the recently emerging topics is solving the problem of minimizing the energy consumption on NoCs [48, 49]. For example, ignoring the possibility of slack recovery is regarded as energy wasting. The energy consumption is also affected by the frequency scaling of the cores and routers. By increasing the execution time via decreasing the clock frequencies in cores, the length of the *Makespan* also increases. However, decreasing the clock frequency also decreases the energy consumption. That mechanism leads to a two-variable trade-off decision to be made between Performance and Energy Consumption, on both, cores and routers.

*Distributed DVFS Algorithm at Router and Core Level*: our model is based on per-core and router DVFS with multiple voltage supply networks. We use DVFS on both cores and routers and propose the use of *Non-Minimal Path Adaptive Routing* [2] to balance network traffic. Specifically, the scheduler regularly tunes the operation frequency of the routers and the cores, based on the scheduling-scenario and NoC workload. Also, the scheduler takes into consideration the router and cores utilization in conjunction with tasks slack allowance. By integrating these two techniques, the scheduler generates schedules, that can balance the traffic across the entire NoC and in turn, minimize the total energy consumption. In this work slack and energy are used as input scenarios in the CM, which is covered by the MeS.

*Static meta-scheduling and mapping policy:* Our static-scheduling policies are designed to reduce the energy consumption at two parallel levels: the first level is the mapping of resources (e.g., cores to tasks, routers, and paths to messages) and then at the second scheduling level using *slowdown factors* for frequency tuning in both, cores and routers. In addition, meta-scheduling with managing and injecting dynamic slack for each task to the static scheduling system can create the full range of optimized schedules with respect to energy consumption.

Definitions: the following definitions are used to describe our optimisation model:

$CRS$ … the set of cores, i.e., execution nodes of the platform

$RTR$ ... the set of routers, i.e., message forwarders of the platform

$TSK$ … the set of tasks to be executed

$MSG$ … of all tasks the set of messages to be sent

$MSG_{IN}(t)$ … the set of input messages of a task $t$

$MSG_{OUT}(t)$ … the set of output messages of a task $t$

$et(t)$ … the execution time of a task $t$

$md(m)$ … the message transmission duration of message m from one node to another one. $md(m)$ is proportional to the size of the message $m$. Assuming that there are $hops(m)$ intermediate nodes between the source and destination of $m$, then the total message duration becomes:

$$md_{total}(m) = md(m) \cdot (hops(m) + 1)$$

$t_{inject}(m)$ ... the time at which the sending of a message $m$ started.

$f$ … clock frequency, with $f_{max}$ being the maximum clock frequency and $f_{sel}$ being the selected clock frequency:

$$f_{sel} \leq f_{max}$$

*Energy Consumption:* The energy $E$ is measured in Joule $(J)$ or Watt-seconds $(Ws)$. The energy consumption $E(d)$ for a time duration $d = [t_1; t_2]$ and the power consumption $P(t)$ at time t is as follows [49]:

$$E(d) = \int_{t1}^{t2} P(t)\, dt \quad | \quad d = [t_1; t_2] \quad (1)$$

The power consumption $P$ is equal to the dynamic power $P_{dyn}$ plus the static power $P_{stc}$ [52]:

$$P = P_{dyn} + P_{stc} \quad (2)$$

There have been used analytical and empirical techniques to model $P_{dyn}$ [53]. We also use an analytic method to model $P_{dyn}$. $P_{dyn}$ tends to dominate contribution to power consumption compared to $P_{stc}$ [53]. Thus, in this work we only consider dynamic power and dynamic energy consumption. $P_{dyn}$ can be modelled as:

$$P_{dyn} = A \cdot S \cdot U^2 \cdot f \qquad (3)$$

where $A$ is the activity factor that refers to fraction between 0 and 1, and can express the total capacity of circuits during each cycle charged or discharged, $S$ is the switched capacitance that refers to aggregate load of system that depended on wire lengths on chip, U is the voltage and $f$ is the frequency [53].
Extracted from above equation, the effective switch capacitance $S_{effective}$ is given as:

$$S_{effective} = A \times S \qquad (4)$$

To keep our analytical model within the limits of the solver CPLEX, we consider $S_{effective}$ to be constant one.
With this simplification the power consumption $P_{dyn}$ becomes proportional to the clock frequency $f$ and the square of voltage $U$:

$$P_{dyn} \propto f \cdot U^2 \quad (5)$$

Let $E_{T,dyn}(t)$ denote the dynamic energy consumption of a task $t$ and $E_{M,dyn}(m)$ denote the dynamic energy consumption of transferring a message $m$. With $P_{dyn}$ being the average power consumption, we can model the dynamic energy consumption as follows:

$$E_{T,dyn}(t) = P_{dyn} \cdot et(t) \qquad (6)$$

$$E_{M,dyn}(m) = P_{dyn} \cdot md(m) \qquad (7)$$

where $et(t)$ and $md(m)$ are inversely proportional to the clock frequency:

$$et(t) \propto \frac{1}{f} \quad \wedge \quad md(m) \propto \frac{1}{f} \qquad (8)$$

*Effective Speed:* $et(t)$ of a task $t$ and $md(m)$ of a message $m$ are given at the maximum clock frequency $f_{max}$. However, the effective execution time, denoted as $\overline{et}(t)$, and the effective message duration, denoted as $\overline{md}(m)$, depend on the selected clock frequency $f_{sel}$ and a contextual parameter $C$:

$$\overline{et}(t) = et(t) \cdot \frac{f_{max}}{f_{sel(t)}} + C \quad (9)$$

$$\overline{md}(m) = md(m) \cdot f_{max}/f_{sel(m)} + C \quad (10)$$

This parameter $C$ is a contextual setting delay, which is either zero in case that the clock frequency for the current task or message is the same as before, or otherwise a constant $C_k$, in case that the clock frequency has been changed at the beginning of the current task or message. The parameter $C$ models the fact that changing the clock frequency in a processor takes some time. In our calculations we assume that $C_k = 1ms$.

In DVFS the voltage $U$ varies approximately proportionally with the clock frequency $f$, i.e., $U \propto f$, the power $P_{dyn}$ is thus proportional to the cube of the clock frequency:

$$P_{dyn} \propto f^3 \quad (11)$$

Based on above equation we can derive that the energy $E_{T,dyn}(t)$ of a task $t$ and the energy $E_{m,dyn}(m)$ of a message $m$ are proportional to the cube of the clock frequency multiplied by the execution/communication time:

$$E_{T,dyn}(t) \propto f^3 \cdot et(t) \quad (12)$$

$$E_{m,dyn}(m) \propto f^3 \cdot md(m) \quad (13)$$

*Frequency Co-Efficient:* regarding equation 12, 13 normal energy metrics cannot be completely cover and consider for our metric needs. In the following, we denote the above proportionality factor of the energy consumption as the frequency co-efficient $\mathrm{FE_t}$ of a task and $\mathrm{FE_m}$ of a message respectively. However, the effective frequency co-efficient, tasks denoted as $\overline{FE_T}$, and the effective message frequency co-efficient, denoted as $\overline{FE_m}$. These frequency co-efficients serve for quantifying the energy reductions in the experimental evaluation and the abstraction from technology-specific parameters such as the switched capacitance.

$$FE_T = et(t) \cdot f^3 \ (14)$$

$$\overline{FE_T} = ( \ \ _{max}/f_{sel(t)})^3 \ (15)$$

$$FE_m = md(m) \cdot f^3 (16)$$

$$\overline{FE_m} = \ (m) \cdot \ (f_{max}/f_{sel(m)})^3 \ (17)$$

*Modelling of the Optimisation Problem:* To generate a static-schedule with CPLEX, the system designer needs to describe the problem with decision variables, constants, constraints and an objective function. In this article we provide an extended model of that used in [3, 8]. In the following we present the new respectively modified parameters.

It is essential to model the relationships between physical, logical and application constraints and without topology dependencies and the resulting impact on performance. The system optimization is the process of utilizing these models for searching for optimum or feasible solutions that best match the physical, logical, and application constraints of the implementation. For a given implementation method, the physical constraints characterize architectural aspects, but in our model, these features are free of network topology. e.g., one of the physical constraints facing the implementation of interconnection networks is the available links between two nodes.

*Connectivity constraints:* These are used for defining physical connectivity (e.g., cores, routers, and links).

*Task Allocation and Assignment Constraints:* In our previous work each core can have assigned only one task. In this work each, each core can have assigned multiple tasks. We use the array $ALLOC$ of allocation variables $alloc_i$ to model the allocation of each task $t_i \in TSK$ to one of the cores in $CRS$:

$$ALLOC = \begin{bmatrix} alloc_1 \\ \vdots \\ \vdots \\ alloc_k \end{bmatrix} \in \{1, \dots, |CRS|\}^k$$

$$k = |TSK| \quad (18)$$

*Hop count: $hops(m)$* denotes the number of intermediate visits of a message $m$ along its transmission path from the sender to the receiver task:

$$HOPS = \begin{bmatrix} hops_1 \\ \vdots \\ \vdots \\ hops_n \end{bmatrix} \in \{1, \dots, MaxHop\}^n$$

$$n = |MSG| \quad (19)$$

with $MaxHop$ being maximum permissable number of intermediate hops.

*Message Duration:* $MD$ is a vector with all the message durations $md(m)$ from one hop to the next hop of all the messages $m \in MSG$:

$$MD = \begin{bmatrix} md_1 \\ \vdots \\ \vdots \\ md_n \end{bmatrix} \in \{1, \dots, Max\}^n \quad | \, n = |MSG| \quad (20)$$

*Slow Down Factors*: One of the essential decision variables in our DVFS optimisation model are the *slow-down factors* of tasks and messages. $tsdf(t)$ denotes the slow-down factor of a core for computing task *t* and $msdf(m)$ denotes the slow-down factor for transmitting message *m*. When assuming that the contextual parameter *C* is zero, then $tsdf(t)$ is proportional to the effective execution time $\overline{et}(t)$: $tsdf(t) \propto \overline{et}(t)$, and $msdf(m)$ is proportional to the effective message duration $\overline{md}(m)$: $msdf(t) \propto \overline{md}(m)$. $TSDF$ is the vector with the slow-down factors of all tasks in $TSK$:

$$TSDF = \begin{bmatrix} tsdf_1 \\ \vdots \\ \vdots \\ tsdf_k \end{bmatrix} \in \{tsdf_{min}, \dots, tsdf_{max}\}^k$$

$$k = |TSK|, \qquad tsdf_{min} \geq 1 \quad (21)$$

Analogously, MSDF is the vector with the slow-down factors of all messages in MSG:

$$MSDF = \begin{bmatrix} msdf_1 \\ \vdots \\ \vdots \\ msdf_n \end{bmatrix} \in \{msdf_{min}, \dots, msdf_{max}\}^n$$
$$n = |MSG|, \qquad msdf_{min} \geq 1 \qquad (22)$$

Based on these slow-down factors, the effective execution time $\overline{et}(t)$ of a task $t$ and the effective message duration $\overline{md}(m)$ of a message $m$ can be written as:

$$\overline{et}(t) = et(t) \cdot tsdf(t) + C$$
$$\overline{md}(m) = md(m) \cdot msdf(m) + C \qquad (23)$$

The total effective message duration $\overline{md}_{total}(m)$ of a message $m$ from sender to receiver becomes:

$$\overline{md}_{total}(m) = \overline{md}(m) \cdot (hops(m) + 1) \qquad (24)$$

*Task Dependency Constraints:* A task dependency between two tasks $t_k$ and $t_l$ is given when the input message of task $t_l$ depends on the output message of task $t_k$. For example, in the *Figure 7* the task $T4$ depends on the tasks $T0, T1, T2$, and $T3$.

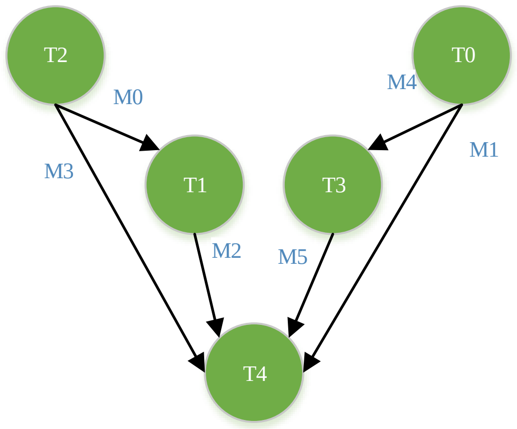


*Figure 7. Conceptual AM*

The following relation of injection times between input messages $MSG_{IN}(t)$ and output messages $MSG_{OUT}(t)$ of any task $t \in TSK$ holds:

$$\forall t \in TSK.\ \forall m_i \in MSG_{IN}(t).\ \forall m_o \in MSG_{OUT}(t) \qquad (25)$$
$$t_{inject}(m_i) + \overline{md}_{total}(m_i) + \overline{et}(t) \leq t_{inject}(m_o) \qquad (26)$$

*Message and task Deadlines:* To support real-time computing, we need to have real-time constraints for tasks and messages. For example, we denote with $D(m)$ the relative deadline of a message $m$. The injection of a message $m$ has to early enough, so that the communication can happen before its deadline $D(m)$. To ensure this, we have to use the following timing constraint for each message $m \in MSG$:

$$t_{inject}(m) + \overline{md}(m) \cdot (MaxHop + 1) \leq D(m) \qquad (27)$$

We also have to make sure that the message $m$ is only read at the destination after its deadline:

$$D(m) \leq t_{read\_dest}(m) \qquad (28)$$

*Ensuring Quadratic Expressions*: We use MIQP (mixed-integer quadric programming) to solve the constraint system of our optimisation model. Hence, any term with three or more variables cannot be used in our optimisation model. However, in our constraints described so far, there are also non-quadratic expressions, for example, $msdf(m)^2 \cdot dm(m) \cdot hops(m)$ with $msdf(m)$ and $hops(m)$ being optimisation variables. To make our expressions quadratic [54, 55], we use some strategy for "not linear" optimisation [56]. While this strategy is typically used for linearisation, we use it for "Quadratisation" [57, 58]. The key principle is to split the optimisation processes into two phases, where in each phase a different variable is treated as constant. The price for this simplification is that it might not always lead to a globally optimal solution.

Figure 8 shows how we use this "Quadratisation" of the constraint system of our model. In the first step we optimise with the model with the slow-down factors set to constant one, i.e. with maximum frequency. From the obtained result we extract the number of hops ($hops(m)$) and now use these values as constants in a second optimisation run where we instead the flow-down factors use as decision variable. This way we can use CPLEX with an MIQP algorithm.

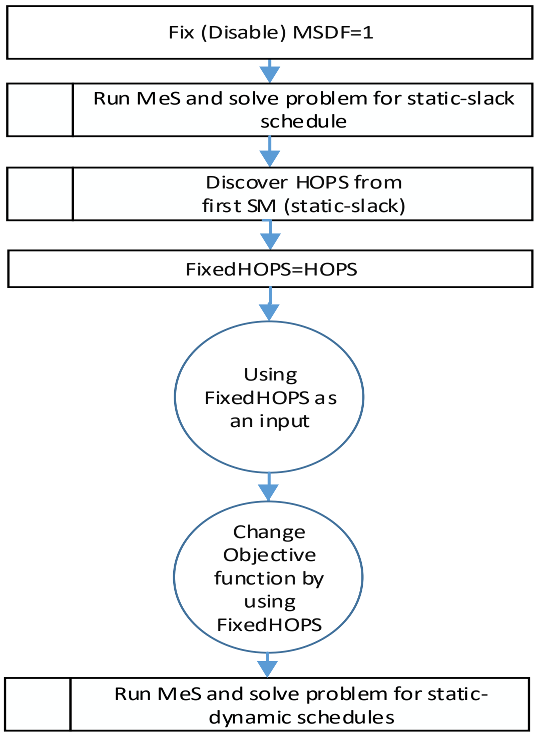

Fix (Disable) MSDF=1
Run MeS and solve problem for static-slack schedule
Discover HOPS from first SM (static-slack)
FixedHOPS=HOPS
Using FixedHOPS as an input
Change Objective function by using FixedHOPS
Run MeS and solve problem for static-dynamic schedules

*Figure 8. "Quadratisation" technique via HOPS*

*The Objective Function*: The objective of our static-scheduling function is to maximize the energy-efficiency for both cores and routers by lowering the clock frequency (c.f in [8] the objective function was to minimise the *Makespan*). In this work, we used a predefined *Makespan* as a constraint variable for controlling schedules timing. The power consumption is minimised by increasing the timing, e.g., task execution times and message

Because CPLEX cannot solve a grade-three objective function, then we used another modelling, which is $md(m)$ and $et(t)$ in scheduling one time is calculated (multiplied) with its slow down factor then on the objective function it used with power two. It models the energy consumption of each task and message (e.g., in whole transmission path), expressed as the task execution time $et(t)$ multiplied to the power of two of the slow down factor $tsdf(t)$ plus the message duration $md(m)$ multiplied to the power of two of the slow-down factor $msdf(m)$ multiplied by the number of hops $(hops(m) + 1)$.

As described in the section about assuring quadratic expressions, the optimisation process is partitioned into two phases.

For the first phase (left bubble of Fig.8) we assume all slow-down factors to be constant:

$$\forall m \in MSG\ .\ msdf_{const}(m) = 1 \qquad (29)$$

Based on that we get a quadratic constraint system with the goal function to be maximised by CPLEX:

$$\forall t \in \mathrm{TSK}\ .\ \forall \mathrm{m} \in MSG\ .$$

$$CP_1(m,t) = et(t) \cdot tsdf(t)^2 + md(m) \cdot msdf_{const}(m)^2 \cdot (hops(m) + 1) \quad (30)$$

$$RES_1 = maximize\left(\sum_{i=1}^{|TSK|} \sum_{j=1}^{|MSG|} \left(CP_1(m_j, t_i)\right)\right) \qquad (31)$$

Based on the optimisation result $RES_1$, the second optimisation phase is done (right bubble of Fig.8). First, the determined hop counts are extracted from the intermediate result $RES_1$:

$$\forall \mathrm{m} \in MSG\ .\ hops_{const}(m)\ is\ extracted\ from\ RES_1$$

Then, the final optimisation result including the slow-down factors is calculated by using the constant values of the hop counts from the first solution:

$$\forall t \in \mathrm{TSK}\ .\ \forall \mathrm{m} \in MSG\ .$$

$$CP_2(m,t) = et(t) \cdot tsdf(t)^2 + md(m) \cdot msdf(m)^2 \cdot (hops_{const}(m) + 1) \quad (32)$$

$$RES_2 = maximize\ (\sum_{i=1}^{|TSK|} \sum_{j=1}^{|MSG|} \left(CP_2(m_j, t_i)\right)) \qquad (33)$$

## V. Implementation of MeS

We extended the MeS for scenario-based scheduling and supporting multi-task on multi-cores, also a concrete power model with discrete frequencies on cores and routers. In this work compare to previous works [3, 6, 7] we are improving new several strategies in different algorithm and methods. In our first strategy, we use a deadline, list of tasks and messages to create schedules concerning the dependencies from the corresponding task and messages. In our second strategy, the CPLEX can select the best slowdown-factor from the set a frequency level with which the original tasks and messages should run. Thus, we use the mapping WCET of all original tasks and message duration as it is and change only the runtime of them by using the selected slowdown factor level. In the third strategy, we worked to used dynamic-slack for efficient *Makespan* in MeS level; then it uses generated new application-model as an input data in the scheduler tool. In the fifth strategy, we let CPLEX deciding and optimizing where is the best location (core) for tasks regarding deadlines, dependencies, slow-down factor and in/out messages. Thus, multi-tasks can run on a core but not all tasks in one core. Finally, the execution times of tasks and injection times of messages are corrected according to the dependencies given by the application model.

*Schedule model tree:* MeS stores all generated scenario-based schedules in a binary-tree structure, which we visualize with our visualizer tool MeSViz [6].

*Schema Modelling:* MeS is a novel flexible architecture for scenario-based scheduling on adaptive time-triggered systems. To achieving this plans in embedded systems era, MeS is modelled conceptually in several structures. Figure 9 presents how the system designer is using the schema method. This method has valuable benefits to save time and cost which it can very fast and easy evaluate every input files via a schema-editor by general schema model as a reference model.

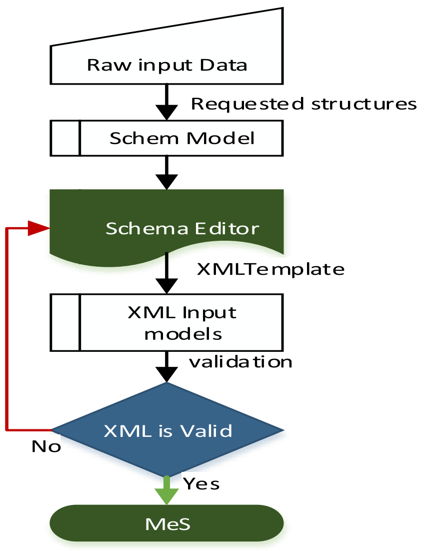


*Figure 9. Schema technique for standard data structure modelling*

Table 1 presented raw data model which is using a regular input data.

*Table 1. Sample raw input data before forming in XML format*

| Application Data | |
|---|---|
| Task ID=0 ET=50energy=[1,100]<br>Task ID=1 ET=60energy=[1,100]<br>Task ID=2 ET=42energy=[1,100]<br>Task ID=3 ET=80energy=[1,100]<br>Task ID=4 ET=90energy=[1,100] | Msg. ID=0 from job#2 to job#1<br>Msg. ID=1 from job#0 to job#4<br>Msg. ID=2 from job#1 to job#3<br>Msg. ID=3 from job#2 to job#3<br>Msg. ID=4 from job#0 to job#3<br>Msg. ID=5 from job#3 to job#4 |
| Physical Data | |
| Router ID5 energy=[0,100]<br>Router ID6 energy=[0,100]<br>Endsystem ID0 energy=[0,100]<br>Endsystem ID1 energy=[25,100] | Link ID=0 from node#0 to node#5<br>Link ID=1 from node#1 to node#5<br>Link ID=2 from node#2 to node#5<br>Link ID=3 from node#3 to node#6 |

| | |
|---|---|
| Endsystem ID2 energy=[50,100]<br>Endsystem ID3 energy=[0,100]<br>Endsystem ID4 energy=[0,100] | Link ID=4 from node#4 to node#6<br>Link ID=5 from node#5 to node#6 |

All input information (e.g., Table 1) is modelled in the schema format via the Oxygen XML editor to describe the structure of XML documents. This rule helps us to prepare the standard and flexible standard building blocks of an XML document in all MeS generations. Figure 10 is a sample of standardized input XML for using in MeS, which is validated via the schema-editor.

```
<ApplicationModel>
<job ID="0" WCET="2" min_energy="1" max_energy="100" deadline="1100"/>
<job ID="1" WCET="4" min_energy="1" max_energy="100" deadline="1600"/>
<job ID="2" WCET="6" min_energy="1" max_energy="100" deadline="1305"/>
<message ID="0" from="0" to="1" size="10" min_energy="1" max_energy="100" deadline="1185"/>
<message ID="1" from="1" to="2" size="10" min_energy="1" max_energy="100" deadline="1185"/>
<message ID="6" from="0" to="5" size="10" min_energy="1" max_energy="100" deadline="1185"/>
</ApplicationModel>
<PlatformModel>
<node ID="0" Type="switch" min_energy="0" max_energy="100"/>
<node ID="1" Type="switch" min_energy="0" max_energy="100"/>
<node ID="2" Type="switch" min_energy="0" max_energy="100"/>
<node ID="3" Type="endsystem" min_energy="0" max_energy="100"/>
<node ID="4" Type="endsystem" min_energy="25" max_energy="100"/>
<link ID="0" from="0" to="1"/>
<link ID="1" from="1" to="2"/>
<link ID="2" from="3" to="0"/>
<link ID="3" from="4" to="1"/>
</PlatformModel>
<ContextModel>
<SlackEvent job="0" NewExecutionTime="50"/>
<SlackEvent job="1" NewExecutionTime="50"/>
<FaultEvent type="crash">
<NodeFault NodeId="0"/>
</FaultEvent>
<FaultEvent type="crash">
<NodeFault NodeId="1"/>
</FaultEvent>
</ContextModel>
```

*Figure 10. Standardized input XML sample*

When the processing of the input models in XML format is done, they are stored internally in MeS as data structure. Regarding defined scenarios in CM, every generated schedule and depended SM, which are stored in an SM class, will be added as a node into the SM tree.

Figure 11 represents a conceptual model of a total process for a scenario-based meta-scheduling technique. It is including input, output data (e.g., XML, gantt map, Graphviz data) and their (e.g., XML parser, MeSViz) processes, data structures (e.g., AM, PM and CM classes, SM's tree), manager and controllers (e.g., event, undo, calendar, Tabu, scheduler) and scheduling processes (e., SM tree creator, scheduler).

*Slack recovery technique:* We use slack time of tasks to increase energy-efficiency in NoCs by optimising the system's timing. When tasks are running in a system, two types of slacks will occurs $Static - Slack\ (SS)$ and $Dynamic - Slack\ (DS)$ as shown in *Figure 12*. Static-slack is the time gap of dependent tasks in different cores, between the end of the first and the start of the second tasks, e.g., static slack $SS0$ between $T1$ and $T2$, and $SS1$ between $T3$ and $T0$, and $SS3$ between T6 and T5 SS3. Other $SS$ examples include two independent tasks on the same core, e.g., $SS2$ occurs between $T6$ and $T4$ on core 4. Dynamic-slack occurs when a task finishes earlier than its WCET, e.g., dynamic slack $DS0$ in $T1$, $DS1$ in $T3$, $DS2$ in $T6$, and $DS3$ in $T4$.

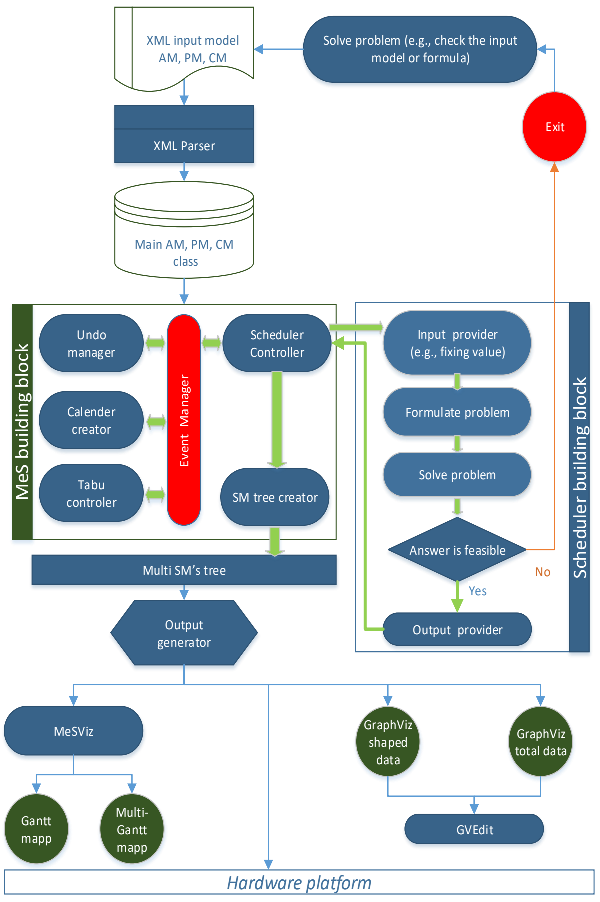
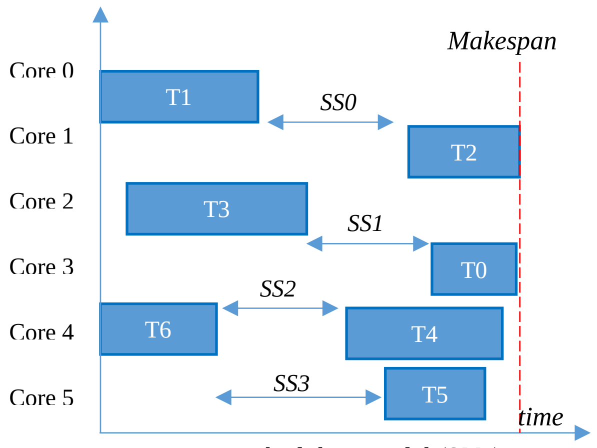


*Figure 11. Conceptual Model of the MeS Tool*

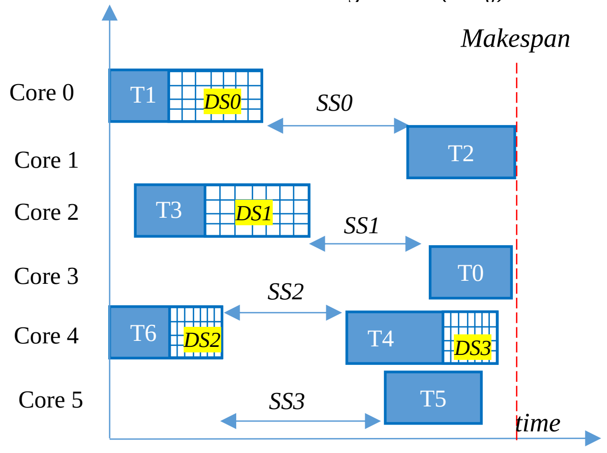
To solve the meta-scheduling problem, e.g., energy optimisation based on slack time, we use the MeS tool, as shown in *Figure 11*. MeS initially starts with generating a static-schedule with scheduling $SM_0$. To take the DS into account, the Event-Manager reduces for each task of the AM its WCET by its slack time, resulting in a new shorter WCET.

This consideration of the DS is marked as an event and injected by Calendar Creator into the calendar table. The Event Manager sends the event type, e.g., slack, and timing to the Scheduler Controller. Depending on the timing of the calendar event, the Scheduler Controller will freeze or unfreeze the corresponding task in the *Tabu* Controller. What the *Tabu* Controller basically does, is to freeze the partial schedule up to the event, and leave the rest to be completed by the scheduler. Afterwards, the AM and PM will be sent to the scheduler to create new schedules by completing the frozen incomplete schedule. The SM Tree Creator generates the scheduling tree (multi SM tree), and depending whether events are taken or not, they are either stored on the right or left side of this tree.

Finally, when every event was taken, and the calendar becomes empty, then MeS calls the Output Generator to generate two output files in Graphviz format: the shaped data and the total data. The total data file includes all the controller parameters



*a. Static scheduling model ($SM_0$)*



*b. Dynamic scheduling model*

*Figure 12. a. Stack-Slack (SS) and b. Dynamic-Slack (DS) sample*

stored in the nodes of the scheduling graph. The shaped data file is a cleaned up version of the SM tree including only the real scheduling nodes, but without the controller parameters. *MeSViz* [6] is used to visualise single schedules as a Gantt map and events depending on schedules as a multi-Gantt map in different graphical format.

*MeS* works with multiple scheduling models organised in tree form, to switch between schedules based on dynamic events. Thus, we introduce $SSM$ as the set of all scheduling models. However, among all the scheduling models $sm \in$ SSM, the one which uses static slack is denoted as $SM_0$. All the other scheduling models SSM/$\{SM_0\}$ use dynamic slack.

MeSViz is used to calculate $FE(sm)$ of each scheduling model.

The energy $FE_C(c, sm)$ consumed at any core $c \in$ CRS for all of its tasks can be calculated as follows:

$$\forall sm \in \text{SSM}.\ \forall c \in \text{CRS}.\ FE_C(c,sm) = \sum_{t \in \text{TSK(c,sm)}} et(t) \cdot \left(\frac{f_{max}}{tsdf(t)}\right)^3 \quad (34)$$

where $TSK(c, sm)$ is defined as the set of tasks mapped to a core $c$ in scheduling model $sm$:

$$TSK(c,sm) = \{t \in TSK \mid alloc_t(sm) = c\}$$

The total energy consumption of all the cores $FE_C(sm)$ can then be calculated as:

$$FE_C(sm) = \sum_{c \in CRS} FE_C(c,sm) \qquad (35)$$

The energy $FE_R(r, sm)$ consumed at any router $r \in$ RTR for all its messages can be calculated as follows:

$$\forall sm \in \text{SSM}.\ \forall r \in RTR.$$

$$FE_R(r,sm) = \sum_{m \in MSG(r,sm)} md(m) \cdot \left(\frac{fmax}{msdf(m)}\right)^3 \qquad (36)$$

where $MSG(r, sm)$ is defined as the set of all the messages going through router $r$ in scheduling model $sm$. The total energy consumption of all the routers $FE_R(sm)$ can then be calculated as:

$$FE_R(sm) = \sum_{r \in RTR} FE_R(r,sm) \qquad (37)$$

In every calculations we assumed $f_{max} = 1$.
The whole energy consumption of a system for a concrete scheduling model $sm$, $FE(sm)$, is calculated from the energy consumption of all the routers and cores as follows:

$$FE(sm) = FE_C(sm) + FE_R(sm) \qquad (38)$$

The average $FE$ energy consumption $FE_{C,avg,dyn}$ of all cores for all scheduling models with dynamic slack can be calculated as follows:

$$FE_{C,avg,dyn} = \frac{1}{(|SSM|-1)} \cdot \sum_{sm \in (\text{SSM}/\{SM_0\})} \sum_{c=0}^{|CRS-1|} FE_C(c,sm) \ (39)$$

The average energy consumption $FE_{R,avg,dyn}$ of all routers for all scheduling models with dynamic slack can be calculated as follows:

$$FE_{R,avg,dyn} = \frac{1}{(|SSM|-1)} \cdot \sum_{sm \in (\text{SSM}/\{SM_0\})} \sum_{r=0}^{|RTR-1|} FE_R(r,sm) \ (40)$$

The average $FE$ energy consumption for all scheduling models with dynamic slack can be calculated as follows:

$$FE_{avg,dyn} = FE_{R,avg,dyn} + FE_{C,avg,dyn} \ (41)$$

MeSViz compares the $FE$ reduction of cores ($ReFC_C$), routers ($ReFE_R$), and combined between $SM_0$, which uses static slack, and the average $FE$ all the other scheduling models with dynamic slack $SM_x$. The relative $FE$ reduction of the average $FE$ of all the scheduling models with dynamic slack $sm \in (SSM/\{SM_0\}$ compared to the $FE$ of the scheduling model with static slack $SM_0$ are computed separately for $SM$ cores $ReFE_C(SM)$, average cores ($ReFE_C$), $SM$ routers $ReFE_R(SM)$, as follows ($ReFE_R$), each $SM$ ($ReFE_{sm}$) and combined ($ReFE$):

$$ReFE_C = \frac{FE_C(SM_0) - FE_{C,avg,dyn}}{FE_C(SM_0)} \cdot 100\% \ (42)$$

$$ReFE_R = \frac{FE_R(SM_0) - FE_{R,avg,dyn}}{FE_R(SM_0)} \cdot 100\% \ (43)$$

$$ReE_{sm} = \frac{\text{FE}(SM_0) - \text{FE}(SM)}{FE(SM_0)} \cdot 100\% \ (44)$$

$$ReFE = \frac{FE(SM_0) - FE_{avg,dyn}}{FE(SM_0)} \cdot 100\% \ (45)$$

*Makespan and slack*: The Makespan $mks(sm)$ of a scheduling model $sm$ is the duration from the start of the first task in sm till the completion of the last task in $sm$. A useful scheduling optimisation is to reduce the Makespan, for example, to make it fit into the time slot of a time-triggered system. MeS uses the occurrence of dynamic-slack to reduce the Makespan by shifting tasks.

With this method the occurrence of dynamic slack shifts the system into a schedule with shorter Makespan by going down along an edge of the SM tree. $SM_0$ sits at the top of the SM tree, thus all other schedules, which have dynamic slack, will have a lower Makespan:

$$\forall sm \in \mathrm{SSM}/\{SM_0\}.\ mks(SM_0) > mks(sm) \quad (46)$$

Given a scheduling model $sm$, by optimising the Makespan we obtain an optimised scheduling model $sm_{opt}$. The achieved saving of Makespan time ($SavT$) is given as:

$$SavT = mks(sm) - mks(sm_{opt}) \quad (47)$$

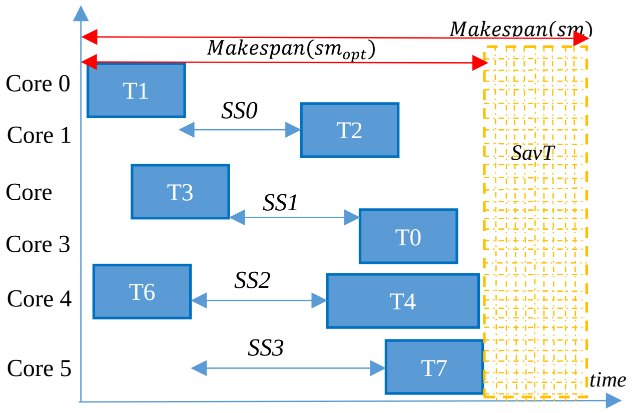


Figure 13. Usage of dynamic slack to educe Makespan

Given the scheduling model shown in Figure *12*.b, the scheduling model $sm_{opt}$ resulting from the Makespan optimisation based on dynamic-slack deployment is shown in Figure 13.

With this optimisation method we aim to minimise energy consumption by reducing Makespan so that the system can go sooner into idle mode.

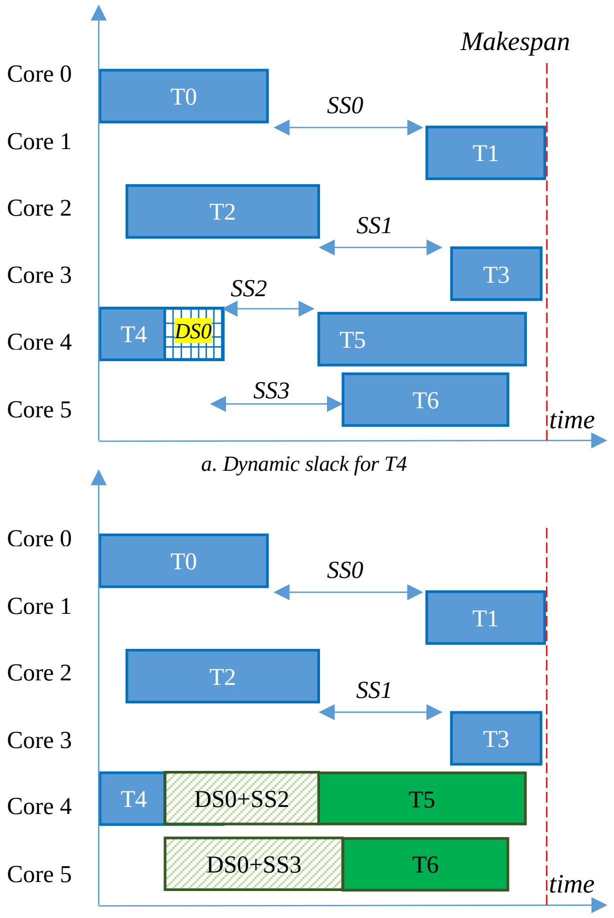
*Power consumption and slack*: With this method we aim to minimise energy consumption to deploying dynamic slack to slow down the components, while still preserving its Makespan. That means we preserve the finish time of the schedule, but use the extra time to the remaining jobs with a lower frequency, thus saving energy since a lower frequency means a lower power consumption. In this method when dynamic slack occurs the next tasks will not shift, but it will run with an increased execution time due to the lower frequency. Thereupon, energy-efficiency will be increased.

Given the scheduling models shown in Figure 14, the scheduling model Figure 14.b resulting from the power consumption optimisation based on dynamic-slack deployment is shown in Figure 14.a. In this example, tasks $T5$ and $T6$ depend on $T4$; when $DS0$ occurred for $T4$, then this free slot and static slacks are used to achieve the maximum slowdown factor by expanding and increasing the execution time of via the frequency of the allocated core(s). The maximum time extending for $T5$ is equal to:

$$Max(et_{t_5}) = et(t_5) + DS_0 + SS_2 \quad (48)$$

Also the optimal slow-down factor of $T5$ is proportional to $Max(et_{t_5})$:

$$et(t_5) \cdot TSDF(t_5) \leq Max(et_{t_5}) \quad (49)$$



b. Adding Dynamic slack and static slack to next tasks

Figure 14. Usage of Slowdown factor regarding Dynamic-Slack

## VI. Example Scenarios And Results

This section presents the visualized results of usage dynamic-slack method in MeS for energy-efficient with linearized *MIQP* model. Hence, in this section, we examine the importance of methods and techniques which are discussed in IV and V. These experiment and related results can answer the question why we require meta-scheduling and dynamic-slack technique and special energy reduction schemes for both cores and routers. Tools were run on a virtual cluster machine with 12 cores of an $Intel\ Xeon\ E5-2450$ 2.2 $GHz$ $and$ 60$GB$ RAM on $Ubuntu$ 14.04.5 ($GNU$ / $Linux$ 3.13.0 − 100 − $generic$ $x$86_64).

*Input models:* The input data for designed use case are shown in Table *2* and Figure 15 shows the AM and Figure 16 shows the PM of the case study. In the AM, $T2$ and $T0$ are starting tasks and T4 is the last task. In PM, $ES2$ with links $L4$ and $L6$ is connected to two hops $H1$ and $H2$. The connection of $ES2$ via two different links is done to provide better reliability [18] of the system. This example shows how MeS can be used to model reliability for safety-critical systems.

*Table 2. MeS input constant*

| Input Model | Input Name | ID | Data |
|---|---|---|---|
| AM | Task | 0 | WCET=2 |
| | | 1 | WCET=4 |
| | | 2 | WCET=6 |
| | | 3 | WCET=8 |
| | | 4 | WCET=10 |
| | Message | Quantity=6 ID start=0 | |
| | TSDF | All Task: min =1 & max =100 | |
| | MSDF | All Messages : min =1 & max =100 | |
| | SlackEvent | All Task= 50% | |
| | Hop (Router) | Quantity=3 ID start=0 | |
| PM | Core | Quantity=5 ID start=6 | |
| | Link | Quantity=9 ID start=0 | |

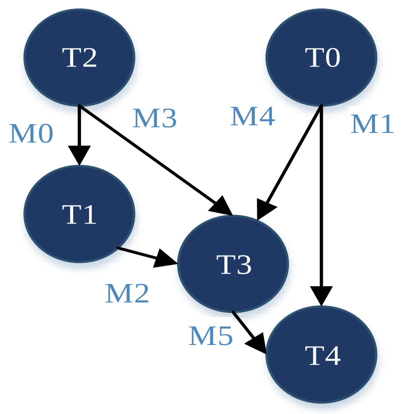


*Figure 15. The AM of the case study*

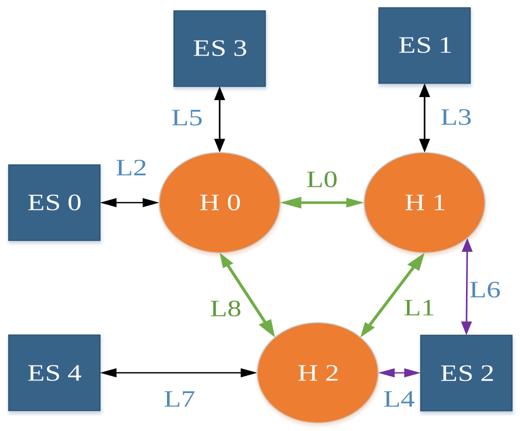
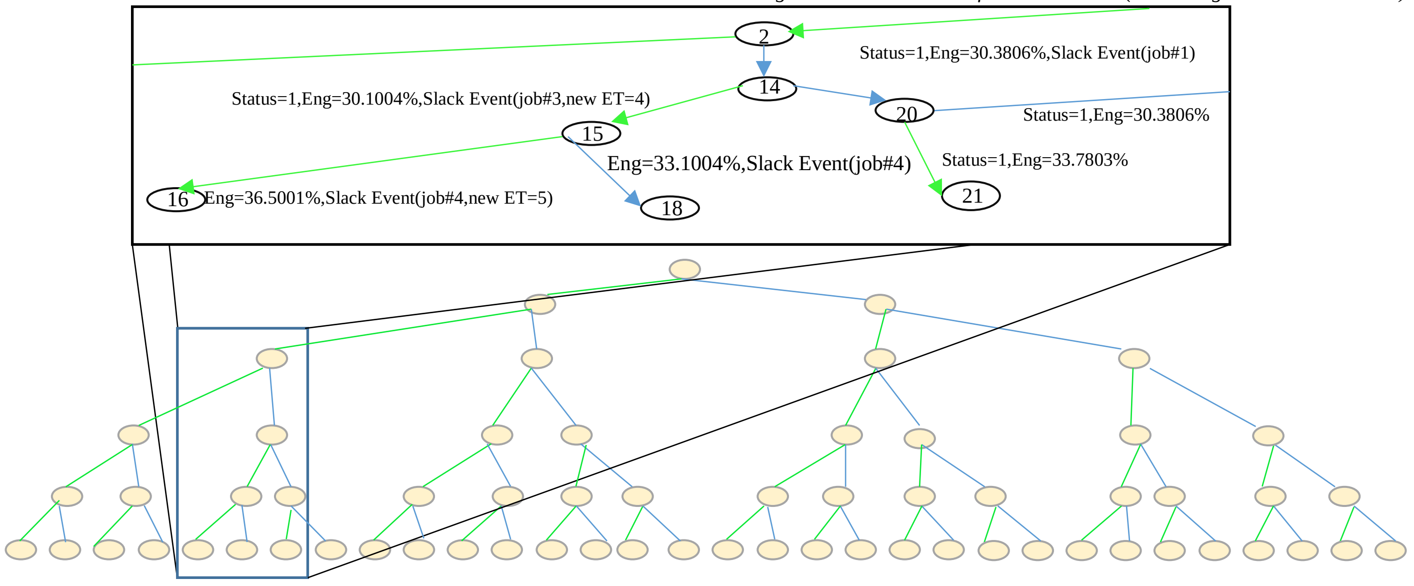


*Figure 16. The PM of the case study*

*Output results:* MeS at 200 seconds generated 93 schedules for the dynamic-slack scenario SSM/$\{SM_0\}$ and one schedule for the static-slack $SM_0$. GVEdit ($GraphViz\ ver.$ 1.02 [59]) creates a graph map from MeSViz output with 94 $SMs$. Figure 18. Schedule tree with 94 schedules (created via MeSViz and GVEdit from Figure 17) shows the schedule tree, where each node contains an SM identifier, each edge represents the schedule status (e.g., $Status = 0$...invalid and $Status = 1$...valid), the energy reduction value, occurred events (e.g., slack), task identifier, and new execution time ET. For example, Figure 17 shows some source code data (created by MeSViz) for creating Figure 18. Schedule tree with 94 schedules (created via MeSViz and GVEdit from Figure 17).

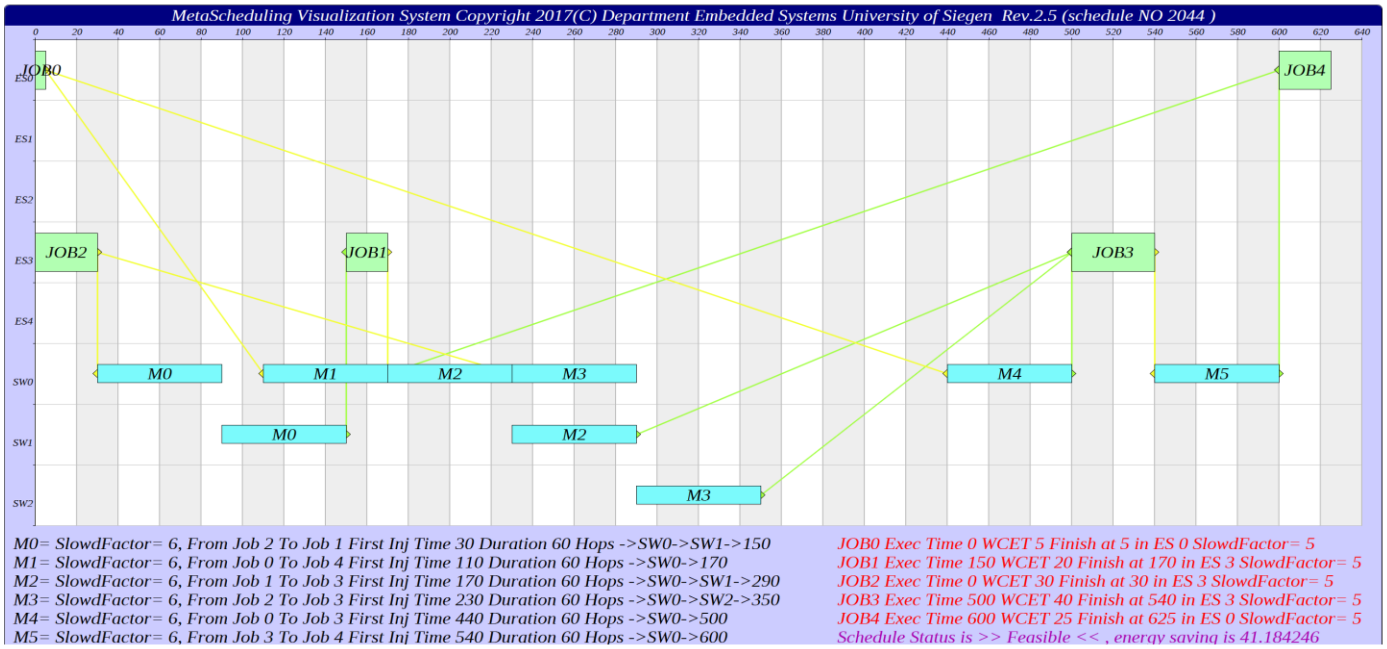
```
14−> 15 [label = "Status = 1, Eng = 33.1004%,
                Slack Event (Job #3, new ET = 4)"color
                = green] ;
15−> 16 [label = "Status = 1, Eng = 36.5001%,
                Slack Event (Job #4, new ET = 5)"color
                = green] ;
15−> 18 [label = "Status = 1, Eng = 33.1004%,
                Job Finish Event (Job #4)"color = blue] ;
```

*Figure 17. Source Code of the three nodes (has been generated via MeSViz)*



2
Status=1,Eng=30.3806%,Slack Event(job#1)
14
Status=1,Eng=30.1004%,Slack Event(job#3,new ET=4)
20
Status=1,Eng=30.3806%
15
Eng=33.1004%,Slack Event(job#4)
Status=1,Eng=33.7803%
16
Eng=36.5001%,Slack Event(job#4,new ET=5)
18
21

*Figure 18. Schedule tree with 94 schedules (created via MeSViz and GVEdit from Figure 17)*


MetaScheduling Visualization System Copyright 2017(C) Department Embedded Systems University of Siegen Rev.2.5 (schedule NO 2044 )

M0= SlowdFactor= 6, From Job 2 To Job 1 First Inj Time 30 Duration 60 Hops ->SW0->SW1->150
M1= SlowdFactor= 6, From Job 0 To Job 4 First Inj Time 110 Duration 60 Hops ->SW0->170
M2= SlowdFactor= 6, From Job 1 To Job 3 First Inj Time 170 Duration 60 Hops ->SW0->SW1->290
M3= SlowdFactor= 6, From Job 2 To Job 3 First Inj Time 230 Duration 60 Hops ->SW0->SW2->350
M4= SlowdFactor= 6, From Job 0 To Job 3 First Inj Time 440 Duration 60 Hops ->SW0->500
M5= SlowdFactor= 6, From Job 3 To Job 4 First Inj Time 540 Duration 60 Hops ->SW0->600

JOB0 Exec Time 0 WCET 5 Finish at 5 in ES 0 SlowdFactor= 5
JOB1 Exec Time 150 WCET 20 Finish at 170 in ES 3 SlowdFactor= 5
JOB2 Exec Time 0 WCET 30 Finish at 30 in ES 3 SlowdFactor= 5
JOB3 Exec Time 500 WCET 40 Finish at 540 in ES 3 SlowdFactor= 5
JOB4 Exec Time 600 WCET 25 Finish at 625 in ES 0 SlowdFactor= 5
Schedule Status is >> Feasible << , energy saving is 41.184246

*Figure 19. (Gantt map of one schedule (name from Figure 11): Dynamic slack SM which generated by MeSViz*

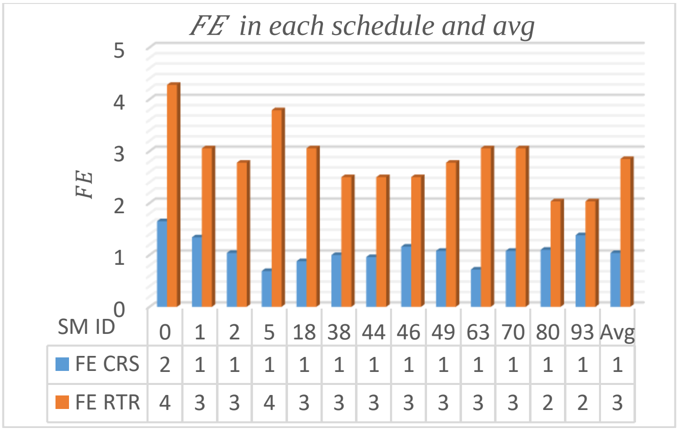
Figure 19 shows the Gantt map for one specific schedule with $SM_{44}$, which has been generated by MeS and visualized via *MeSViz*. Figure 19 includes all the information that is needed for debugging and implementation. For example, it shows that from five cores only two are used, how the tasks and messages are allocated and depend on each other, the values of the *slowdown factor* for each task and message, the scheduled status, the energy reduction rate, the message path from one hope to another hop, and the timing of messages and jobs.

*Discussion:* In Table 3 the results of fourteen samples of collected data from $SM$ outputs are shown. The schedule $\mathrm{SM}_0$is used for static slack time, the schedule $\mathrm{SM}_1$ to $\mathrm{SM}_{93}$ are used for dynamic slack time. As given in the results, schedule $\mathrm{SM}_5$ has lowest $FE$ energy reduction (saving) and schedule $\mathrm{SM}_{80}$ has highest $FE$ energy reduction.

*Table 3. General results of $FE$ for model example*

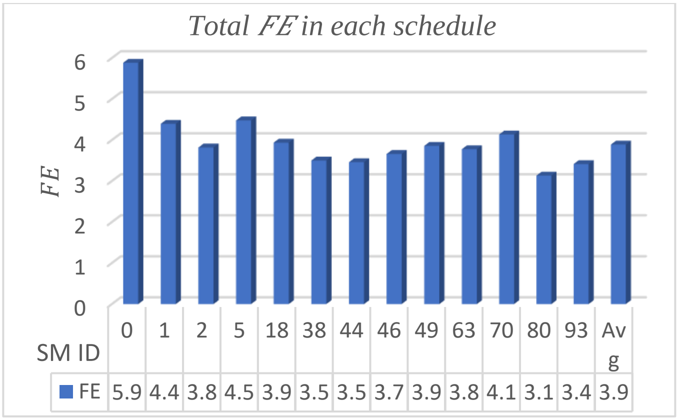
| SM ID | FE CRS | FE RTR | ReFE Core | ReFE RTR | FE | ReFE Total |
|---|---|---|---|---|---|---|
| 0 | ↑ 1.65 | ↑ 4.2778 | ↓ 0.0000% | ↓ 0.0000% | ↑ 5.88278 | ↓ 0.0000% |
| 1 | ↑ 1.34 | ↑ 3.0556 | ↓ 16.5109% | ↓ 28.5714% | ↑ 4.3956 | ↓ 25.2802% |
| 2 | ↓ 1.04 | ↓ 2.7778 | ↓ 35.2025% | ↑ 35.0649% | ↓ 3.8178 | ↑ 35.1021% |
| 5 | ↓ 0.69 | ↑ 3.7889 | ↑ 57.0093% | ↓ 11.4286% | ↑ 4.4789 | ↓ 23.8642% |
| 18 | ↓ 0.88 | ↑ 3.0556 | ↑ 45.1713% | ↓ 28.5714% | ↑ 3.9356 | ↓ 33.0997% |
| 38 | ↓ 1 | ↓ 2.5 | ↑ 37.6947% | ↑ 41.5585% | ↓ 3.5 | ↑ 40.5043% |
| 44 | ↓ 0.96 | ↓ 2.5 | ↑ 40.1869% | ↑ 41.5585% | ↓ 3.46 | ↑ 41.1843% |
| 46 | ↑ 1.16 | ↓ 2.5 | ↓ 27.7259% | ↑ 41.5585% | ↓ 3.66 | ↑ 37.7845% |
| 49 | ↓ 1.08 | ↓ 2.7778 | ↓ 32.7103% | ↑ 35.0644% | ↓ 3.8578 | ↑ 34.4222% |
| 63 | ↓ 0.72 | ↑ 3.0556 | ↑ 55.1402% | ↓ 28.5714% | ↓ 3.7756 | ↑ 35.8195% |
| 70 | ↓ 1.08 | ↑ 3.0556 | ↓ 32.7103% | ↓ 28.5714% | ↑ 4.1356 | ↓ 29.6999% |
| 80 | ↓ 1.1 | ↓ 2.0333 | ↓ 31.4642% | ↑ 52.4676% | ↓ 3.1333 | ↑ 46.7378% |
| 93 | ↑ 1.38 | ↓ 2.0333 | ↓ 14.0187% | ↑ 52.4676% | ↓ 3.4133 | ↑ 41.9781% |
| **Avg** | **1.03903** | **2.85036** | **35.2629%** | **33.3682%** | **3.88939** | **33.8852%** |

The $FE$ for static $\mathrm{SM}_0$ and dynamic slack modes $SSM/SM_0$ results, $FE$ reductions $ReFE(SSM/SM_0)$, average of $FE$ (for cores and routers) and $ReFE_{total}$, are compared to the $\mathrm{SM}_0$ are shown in Figure 20, Figure 21, Figure 22 and Figure 23.



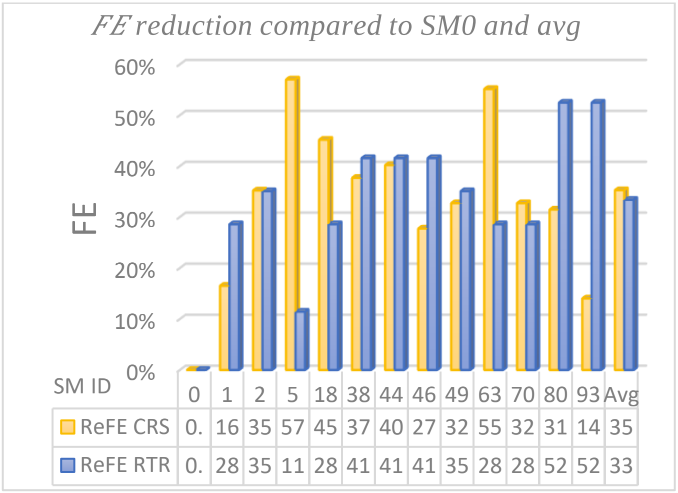
| SM ID | 0 | 1 | 2 | 5 | 18 | 38 | 44 | 46 | 49 | 63 | 70 | 80 | 93 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 🟦 FE CRS | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 🟧 FE RTR | 4 | 3 | 3 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 3 |

*Figure 20. $FE_C(sm), FE_R(sm)$ results for cores and routers and average component frequency*



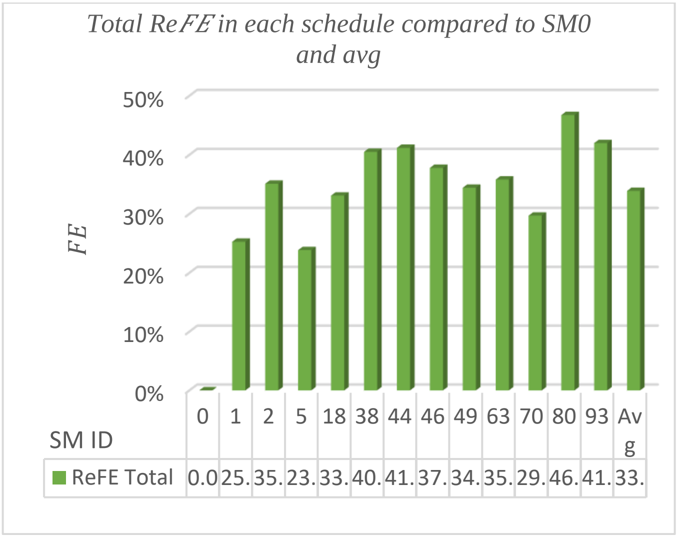
| SM ID | 0 | 1 | 2 | 5 | 18 | 38 | 44 | 46 | 49 | 63 | 70 | 80 | 93 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 🟦 FE | 5.9 | 4.4 | 3.8 | 4.5 | 3.9 | 3.5 | 3.5 | 3.7 | 3.9 | 3.8 | 4.1 | 3.1 | 3.4 | 3.9 |

*Figure 21. $FE(sm)$ results and average*



| SM ID | 0 | 1 | 2 | 5 | 18 | 38 | 44 | 46 | 49 | 63 | 70 | 80 | 93 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 🟨 ReFE CRS | 0. | 16 | 35 | 57 | 45 | 37 | 40 | 27 | 32 | 55 | 32 | 31 | 14 | 35 |
| 🟦 ReFE RTR | 0. | 28 | 35 | 11 | 28 | 41 | 41 | 41 | 35 | 28 | 28 | 52 | 52 | 33 |

*Figure 22.* $FE$ *results for cores* $ReFE_C$ *and routers* $ReFE_R$ *compare to* $SM_0$ *and average*



| SM ID | 0 | 1 | 2 | 5 | 18 | 38 | 44 | 46 | 49 | 63 | 70 | 80 | 93 | Av g |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 🟩 ReFE Total | 0.0 | 25. | 35. | 23. | 33. | 40. | 41. | 37. | 34. | 35. | 29. | 46. | 41. | 33. |

*Figure 23. Total* $ReFE(sm)$ *in each schedule compare to* $SM_0$ *and average*

These results confirm our theory that compared to static slack time, the use of dynamic slack time together with our MeS algorithm can reduce the power consumption of NoCs by using frequency scaling for both cores and routers.

## VII. Conclusion

We have proposed and developed a new scenario-based and energy-efficient meta-scheduling method for NoC-based MPSoCs. Our algorithm minimises the frequency of cores and routers in order to maximise the lifetime of tasks and messages. To do so, a *slowdown factor* is used for each task and message in different scenarios. The power consumption of the NoC is dynamically managed to reduce energy efficiency. Using meta-scheduling, each occurrence of dynamic slack switches the system into a more energy-efficient schedule. The novel optimisation technique of our MeS tool is independent of network design, is expandable, and it can be use to reduce and optimizing the dynamic power consumption in the schedules. MeS is used for our multi-scenario-based (e.g., fault, safety, power-saving) scheduling on adaptive time-triggered systems. The results show that our dynamic slack time consideration and frequency slowdown in MeS, compared to the static slack time, produces a maximum of up to $57\%$ energy reduction for cores and 52.46% for routers. The energy reduction are up to 46.73% in a single schedule and 33.88% energy reduction of NoCs on average.

Despite these useful results, future work is needed to provide additional features like multi-link scheduling, fault-injection and more support for safety-related scenarios. We are going to extend and develop these techniques and methods for more energy-efficient scheduling with reliability and fault-tolerance in the scenario-based domain.

## VIII. References

[1] DW BUSINESS, *BMW increases R&D spending on e-cars, autonomous vehicles*.

[2] J. Yin, P. Zhou, A. Holey, S. S. Sapatnekar, and A. Zhai, “Energy-efficient non-minimal path on-chip interconnection network for heterogeneous systems,” in *ISPLED'12: Proceedings of the international symposium on low power electronics and design*, Redondo Beach, California, USA, 2012, p. 57.

[3] B. Sorkhpour, A. Murshed, and R. Obermaisser, “Meta-scheduling techniques for energy-efficient robust and adaptive time-triggered systems,” in *Knowledge-Based Engineering and Innovation (KBEI), 2017 IEEE 4th International Conference on*, 2017, pp. 143–150.

[4] F. Guan, L. Peng, L. Perneel, H. Fayyad-Kazan, and M. Timmerman, “A Design That Incorporates Adaptive Reservation into Mixed-Criticality Systems,” *Scientific Programming*, vol. 2017, 2017.

[5] Y. Lin, Y.-l. Zhou, S.-t. Fan, and Y.-m. Jia, “Analysis on Time Triggered Flexible Scheduling with Safety-Critical System,” in *Chinese Intelligent Systems Conference*, 2017, pp. 495–504.

[6] B. Sorkhpour and R. Obermaisser, “MeSViz: Visualizing Scenario-based Meta-Schedules for Adaptive Time-Triggered Systems,” in *AmE 2018-Automotive meets Electronics; 9th GMM-Symposium*, 2018, pp. 1–6.

[7] B. Sorkhpour, O. Roman, and Y. Bebawy, Eds., *Optimization of Frequency-Scaling in Time-Triggered Multi-Core Architectures using Scenario-Based Meta-Scheduling*: VDE, 2019.

[8] A. Murshed, R. Obermaisser, H. Ahmadian, and A. Khalifeh, “Scheduling and allocation of time-triggered and event-triggered services for multi-core processors with networks-on-a-chip,” pp. 1424–1431.

[9] F. Pop, C. Dobre, C. Stratan, A. Costan, and V. Cristea, “Dynamic Meta-Scheduling Architecture Based on Monitoring in Distributed Systems,” in *International Conference on Complex, Intelligent and Software Intensive Systems, 2009*, Fukuoka, Japan, 2009, pp. 388–395.

[10] A. Al-Khateeb, A. Rashid, and R. Abdullah, “An enhanced meta-scheduling system for grid computing that considers the job type and priority,” (eng), *Computing : Archives for Scientific Computing*, vol. 94, no. 5, pp. 389–410, http://dx.doi.org/10.1007/s00607-011-0168-6, 2012.

[11] B. Sorkhpour, “Scenario-based meta-scheduling for energy-efficient, robust and adaptive time-triggered multi-core architectures,” Doctoral thesis, Universität Siegen, 2019.

[12] *Green computing: power optimisation of VFI-based real-time multiprocessor dataflow applications (extended version)*: University of Twente, Centre for Telematics and Information Technology (CTIT), 2015.

[13] S. A. H. Abdulsalam, Z. Zong, A. Qasem, and M. Burtscher, *Using the greenup, powerup and speedup metrics to evaluate software energy efficiency*. San Marcos, Texas: Texas State University, 2016.

[14] O. A. Carvalho Junior, S. M. Bruschi, R. H. C. Santana, and M. J. Santana, “Green Cloud Meta-Scheduling,” *Journal of Grid Computing*, vol. 14, no. 1, pp. 109–126, 2016.

[15] H. Castro *et al.*, “Green flexible opportunistic computing with task consolidation and virtualization,” *Cluster Computing*, vol. 16, no. 3, pp. 545–557, 2013.

[16] R. Jejurikar and R. Gupta, “Dynamic slack reclamation with procrastination scheduling in real-time embedded systems,” in *Proceedings of the 42nd annual Design Automation Conference*, 2005, pp. 111–116.

[17] N. Chatterjee, S. Paul, and S. Chattopadhyay, “Task mapping and scheduling for network-on-chip based multi-core platform with transient faults,” *Journal of Systems Architecture*, vol. 83, pp. 34–56, 2018.

[18] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*: Springer Science & Business Media, 2011.

[19] G. Fohler, “Changing operational modes in the context of pre run-time scheduling,” *IEICE transactions on information and systems*, vol. 76, no. 11, pp. 1333–1340, 1993.

[20] R. Obermaisser, Ed., *Time-triggered communication*. Boca Raton, FL: CRC Press, 2012.

[21] Hermann Kopetz, Fellow, IEEE G¨unther Bauer, “The Time-Triggered Architecture,”

[22] J. Theis, G. Fohler, and S. Baruah, “Schedule table generation for time-triggered mixed criticality systems,” *Proc. WMC, RTSS*, pp. 79–84, 2013.

[23] H. Isakovic and R. Grosu, “A Mixed-Criticality Integration in Cyber-Physical Systems: A Heterogeneous Time-Triggered Architecture on a Hybrid SoC Platform,” in *Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications*: IGI Global, 2018, pp. 1153–1178.

[24] H. Isakovic and R. Grosu, “A heterogeneous time-triggered architecture on a hybrid system-on-a-chip platform,” in *Proceedings, 2016 IEEE 25th International Symposium on Industrial Electronics (ISIE): Santa Clara Convention Center, Santa Clara, CA, United States, 08-10 June, 2016*, Santa Clara, CA, USA, 2016, pp. 244–253.

[25] H. F. Sheikh and I. Ahmad, “Simultaneous optimization of performance, energy and temperature for DAG scheduling in multi-core processors,” in *Green Computing Conference (IGCC), 2012 International*, 2012, pp. 1–6.

[26] J. Hu and R. Marculescu, “Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints,” in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, 2004, pp. 234–239.

[27] D. M. Brooks *et al.*, “Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors,” *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000.

[28] S. Prabhu, *Ocin_tsim - A DVFS Aware Simulator for NoC Design Space Exploration and Optimization*. [College Station, Tex.]: [Texas A & M University], 2010.

[29] S. Chai, Y. Li, J. Wang, and C. Wu, “An energy-efficient scheduling algorithm for computation-intensive tasks on NoC-based MPSoCs,” *Journal of Computational Information Systems*, vol. 9, no. 5, pp. 1817–1826, 2013.

[30] D. Li and J. Wu, “Energy-efficient contention-aware application mapping and scheduling on NoC-based MPSoCs,” *Journal of Parallel and Distributed Computing*, vol. 96, pp. 1–11, 2016.

[31] W. Y. Lee, Y. W. Ko, H. Lee, and H. Kim, “Energy-efficient scheduling of a real-time task on dvfs-enabled multi-cores,” in *Proceedings of the 2009 International Conference on Hybrid Information Technology*, 2009, pp. 273–277.

[32] K. Han, J.-J. Lee, J. Lee, W. Lee, and M. Pedram, “TEI-NoC: Optimizing Ultralow Power NoCs Exploiting the Temperature Effect Inversion,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 2, pp. 458–471, 2018.

[33] H. Wang, L.-S. Peh, and S. Malik, “Power-driven design of router microarchitectures in on-chip networks,” in *36th International symposium on microarchitecture*, San Diego, CA, USA, 2003, pp. 105–116.

[34] U. U. Tariq, H. Wu, and S. Abd Ishak, “Energy-Aware Scheduling of Conditional Task Graphs on NoC-Based MPSoCs,” in *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[35] B. D. de Dinechin and A. Graillat, “Network-on-chip service guarantees on the kalray MPPA-256 bostan processor,” in *Proceedings of the 2nd International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems - AISTECS '17*, Stockholm, Sweden, 2017, pp. 35–40.

[36] KALRAY Corporation, *Kalray’s MPPA network-on-chip.* [Online] Available: http://www.kalrayinc.com/portfolio/processors/.

[37] R. Lazimy, “Mixed-integer quadratic programming,” *Mathematical Programming*, vol. 22, no. 1, pp. 332–349, 1982.

[38] IBM, *IBM ILOG CPLEX Optimization Studio CPLEX User’s Manual*: IBM, 1987-2016.

[39] A. Schrijver, *Theory of linear and integer programming*: John Wiley & Sons, 1998.

[40] A. Schrijver, *Theory of linear and integer programming*. Chichester: John Wiley and Sons, 2000.

[41] P. Benner, P. Ezzatti, E. Quintana-Ortí, and A. Remón, “On the Impact of Optimization on the Time-Power-

Energy Balance of Dense Linear Algebra Factorizations," in *Algorithms and Architectures for Parallel Processing*.

[42] A. Majd, G. Sahebi, M. Daneshtalab, and E. Troubitsyna, "Optimizing scheduling for heterogeneous computing systems using combinatorial meta-heuristic solution," in *2017 IEEE SmartWorld: Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI) : 2017 conference proceedings : San Francisco Bay Area, California, USA, August 4-8, 2017*, San Francisco, CA, 2017, pp. 1–8.

[43] A. C. Persya and T. R. G. Nair, "Model based design of super schedulers managing catastrophic scenario in hard real time systems," in *2013 International Conference on Information Communication and Embedded Systems (ICICES 2013)*, Chennai, pp. 1149–1155.

[44] T. Tiendrebeogo, "Prospect of Reduction of the GreenHouse Gas Emission by ICT in Africa," in *e-Infrastructure and e-Services*.

[45] A. Carvalho Junior, M. Bruschi, C. Santana, and J. Santana, "Green Cloud Meta-Scheduling : A Flexible and Automatic Approach," (eng), *Journal of Grid Computing : From Grids to Cloud Federations*, vol. 14, no. 1, pp. 109–126, http://dx.doi.org/10.1007/s10723-015-9333-z, 2016.

[46] H. Jung, H. Oh, and S. Ha, "Multiprocessor scheduling of a multi-mode dataflow graph considering mode transition delay," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 2, p. 37, 2017.

[47] M. Ma and R. Sakellariou, "Reducing Code Size in Scheduling Synchronous Dataflow Graphs on Multicore Systems," in *PARMA-DITAM 2018 proceedings: 9th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures ; 7th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms : January 23, 2018, Manchester, United Kingdom*, Manchester, United Kingdom, 2018, pp. 57–62.

[48] P. Eitschberger, S. Holmbacka, and J. Keller, "Trade-Off Between Performance, Fault Tolerance and Energy Consumption in Duplication-Based Taskgraph Scheduling," in *Architecture of Computing Systems – ARCS 2018*.

[49] P. Eitschberger, "Energy-efficient and Fault-tolerant Scheduling for Manycores and Grids," Fakultät für Mathematik und Informatik, FernUniversität in Hagen, Hagen, 2017.

[50] K. M. Tarplee, R. Friese, A. A. Maciejewski, and H. J. Siegel, "Efficient and Scalable Pareto Front Generation for Energy and Makespan in Heterogeneous Computing Systems," in *Recent Advances in Computational Optimization*.

[51] R. Lent, "Grid Scheduling with Makespan and Energy-Based Goals," *Journal of Grid Computing*, vol. 13, no. 4, pp. 527–546, 2015.

[52] A. Sarwar, "Cmos power consumption and cpd calculation," *Proceeding: Design Considerations for Logic Products*, 1997.

[53] S. Kaxiras and M. Martonosi, "Computer Architecture Techniques for Power-Efficiency," *Synthesis Lectures on Computer Architecture*, vol. 3, no. 1, pp. 1–207, 2008.

[54] P. Bhattacharjee, A. J. Mondal, A. Majumder, and S. K. Metya, "Amplifier design and optimization using Non Linear Programming," in *Proceedings of the 26th International Conference RADIOELEKTRONIKA 2016: April 19-20, 2016, Košice, Slovakia*, Kosice, Slovakia, 2016, pp. 22–27.

[55] D. Kouzoupis, G. Frison, A. Zanelli, and M. Diehl, "Recent Advances in Quadratic Programming Algorithms for Nonlinear Model Predictive Control," *Vietnam Journal of Mathematics*, vol. 46, no. 4, pp. 863–882, 2018.

[56] D. Ahlbom, *Quadratic Programming Modelsin Strategic Sourcing Optimization.* Available: http://www.diva-portal.org/smash/get/diva2:1159097/FULLTEXT01.

[57] H. D. Sherali and C. H. Tuncbilek, "A reformulation-convexification approach for solving nonconvex quadratic programming problems," (eng), *Journal of global optimization : an international journal dealing with theoretical and computational aspects of seeking global optima and their applications in science, management and engineering*, vol. 7 (1995), pp. 1–31, http://dx.doi.org/10.1007/BF01100203, 1995.

[58] D. Axehill, "Applications of integer quadratic programming in control and communication," Institutionen för systemteknik.

[59] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz—open source graph drawing tools," in *International Symposium on Graph Drawing*, 2001, pp. 483–484.