Reliability-aware energy optimization for throughput-constrained applications on MPSoC

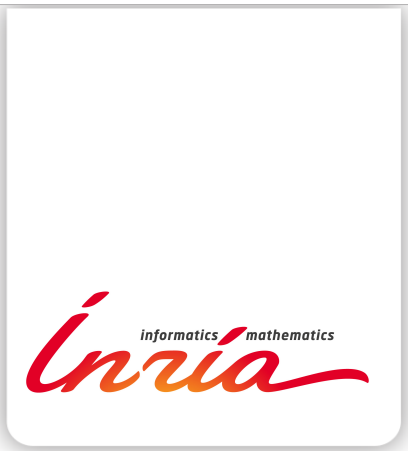
Changjiang Gou, Anne Benoit, Mingsong Chen, Loris Marchal, Tongquan Wei

Reliability-aware energy optimization for throughput-constrained applications on MPSoC

Changjiang Gou, Anne Benoit, Mingsong Chen, Loris Marchal, Tongquan Wei

RESEARCH REPORT
N° 9168
April 2018
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--9168--FR+ENG



Reliability-aware energy optimization for throughput-constrained applications on MPSoC

Changjiang Gou*†, Anne Benoit*‡, Mingsong Chen†,
Loris Marchal*, Tongquan Wei†

Project-Team ROMA

Research Report n° 9168 — April 2018 — 35 pages

Abstract: Multi-Processor System-on-Chip (MPSoC) has emerged as a promising embedded architecture to meet the increasing performance demand of embedded applications. However, due to limited energy budget, it is hard to guarantee that applications on MPSoC can be accomplished on time with a required throughput. The situation becomes even worse for applications with high reliability requirements, since extra energy will be inevitably consumed by task re-executions or duplicated tasks. Based on Dynamic Voltage and Frequency Scaling (DVFS) and task duplication techniques, this paper presents a novel energy-efficient scheduling model, which aims at minimizing the overall energy consumption of MPSoC applications under both throughput and reliability constraints. The problem is shown to be NP-complete, and several polynomial-time heuristics are proposed to tackle this problem. Comprehensive simulations on both synthetic and real application graphs show that our proposed heuristics can meet all the given constraints, while reducing the energy consumption.

Key-words: Scheduling, MPSoC, energy minimization, throughput, reliability.

* Laboratoire LIP, École Normale Supérieure de Lyon & CNRS & Inria, France
† Shanghai Key Lab. of Trustworthy Computing, East China Normal University, China
‡ Georgia Institute of Technology, USA

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Optimisation de la consommation énergétique d'applications MPSoC sous contraintes de fiabilité et de débit

Résumé : Le système multiprocesseur sur puce (MPSoC) est une architecture prometteuse pour répondre à la demande de performance croissante des applications embarquées. Cependant, en raison de leur budget énergétique limité, il est difficile de garantir que les applications sur MPSoC peuvent être accomplies à temps avec un débit requis. La situation devient encore pire pour les applications présentant des exigences de fiabilité élevées, car une énergie supplémentaire sera inévitablement consommée par des ré-exécutions de tâches ou des tâches dupliquées. Basé sur le DVFS (Dynamic Voltage and Frequency Scaling) et la duplication de tâches, cet article présente un nouveau modèle d'ordonnancement, qui vise à minimiser la consommation d'énergie globale des applications MPSoC sous des contraintes de débit et de fiabilité. Le problème est montré NP-complet, et plusieurs heuristiques en temps polynomial sont proposées pour résoudre ce problème. Des simulations complètes sur des graphes d'application tant synthétiques que réels montrent que nos heuristiques peuvent répondre à toutes les contraintes données, tout en réduisant la consommation d'énergie.

Mots-clés : Ordonnancement, MPSoC, consommation énergétique, débit, fiabilité.

1 Introduction

Many smart applications in areas such as Internet of Things (IoT), augmented reality, and robotics, increasingly require high performance on embedded processing platforms. The three main criteria are i) computational performance, expressed as the throughput of the application; ii) reliability, i.e., most data sets must be successfully computed; and iii) energy efficiency. This is mainly because: i) some applications such as audio/video coding or deep learning-based inference are delay-sensitive, hence throughput should be properly guaranteed; ii) emerging safety-critical applications such as self-driving vehicles and tactile internet impose extremely stringent reliability requirements [9]; and iii) devices on which smart applications are running are often battery-operated, hence systems should be energy-efficient.

In order to meet all these design constraints, MPSoC is becoming a new paradigm that enables effective and efficient design of smart applications. By integrating multiple cores together with an interconnection fabric (e.g., Network-on-Chip) as communication backbone, MPSoC (e.g., *OMPA* from Texas Instruments and *NORMADIC* from STMicroelectronics) can be tailored as multiple application-specific processors with high throughputs but low energy consumption [4, 20].

As one of the most effective power management techniques, Dynamic voltage and frequency scaling (DVFS) has been widely used by modern MPSoCs [6]. By properly lowering the processing voltages and frequencies of dedicatedly mapped tasks, DVFS enables smart applications to be carried out with a reduced energy consumption, while ensuring a given throughput. However, scaling down voltages and frequencies of processors generates serious reliability problems. Various phenomena such as high energy cosmic particles and cosmic rays may cause the change of binary values held by transistors within CMOS processors by mistake, resulting in notorious transient faults (i.e., soft errors). Along with the increasing number of transistors integrated on a chip according to Moore's Law, the susceptibility of MPSoC to transient faults will increase by several orders of magnitude [23]. In other words, the probability of incorrect computation or system crashes will become higher due to soft errors.

To mitigate the impact of soft errors, checkpointing and task replication techniques have been widely used to ensure system reliability [11, 16]. Tasks can be replicated if they do not have an internal state, this increases their reliability as it is extremely unlikely to have errors on two or more copies. Although checkpointing and task replication techniques are promising on enhancing the system reliability, frequent utilization of such fault-tolerance mechanisms is very time or resource consuming, which will in turn cost ex-

tra energy and degrade the system throughput. Clearly, the MPSoC design objectives (i.e., energy, reliability and throughput) are three contradictory requirements when we need to decide the voltage and frequency level assignments for tasks. Although there exist dozens of approaches that can effectively handle the trade-off between energy and reliability issues, few of them consider the throughput requirement in addition, see Section 2. Hence, given throughput and reliability constraints, how to achieve a fault-tolerant schedule that minimizes the energy consumption for a specific DVFS enabled MPSoC platform is becoming a major challenge for designers of smart applications.

To address the above problem, this paper proposes a novel scheduling approach that can generate energy-efficient and soft error resilient mappings for smart applications on a given MPSoC platform. It makes following three major contributions:

1. We propose a novel model that can formally express both performance and reliability constraints for mapping applications on MPSoCs, by bounding the expected period (for performance) and the probability of exceeding the target expected period (for reliability).

2. We prove that without performance and reliability constraints, the problem is polynomially tractable, whereas adding both constraints results in an NP-complete problem.

3. We design and evaluate novel task scheduling heuristics for reliability-aware energy optimization on MPSoCs, which enforce the constraints and aim at minimizing the energy cosumption.

The reminder of this paper is organized as follows. Related work is discussed in Section 2. Then, we formalize the application model and optimization problem in Section 3. Section 4 studies the complexity of the problem variants, and in particular proves that the complete version of the problem is NP-complete. To quickly achieve efficient mappings, Section 5 presents the details of our heuristic approaches. Section 6 conducts the evaluation of our approaches on both real and synthetic applications. Finally, Section 7 concludes and provides directions for future work.

2 Related work

MPSoCs have been deployed in various embedded applications such as image processing, process control, and autonomous navigations. Typical MPSoC such as AsAp2 [18] consists of many identical processors with independent

clock domains. They are especially designed for embedded multimedia applications, and featured as high energy efficiency and performance and easy to program [18]. Throughput maximization problem has been a subject of continuing interest as the demands for MPSoC-enabled high performance computing drastically increase. Zhang et al. [22] optimized the throughput in disruption tolerant networks via distributed workload dissemination, and designed a centralized polynomial-time dissemination algorithm based on the shortest delay tree. Li et al. [14] specifically considered stochastic characteristics of task execution time to tradeoff between schedule length (i.e., throughput) and energy consumption. A novel Monte Carlo based task scheduling is developed to maximize the expected throughput without incurring a prohibitively high time overhead [24]. Albers et al. [1] introduced an online algorithm to further maximize throughput with parallel schedules. However, reliability issues are not considered in these works.

Reliability can be achieved by reserving some CPU time for re-executing faulty tasks due to soft errors [23]. In [15], the authors present a representative set of techniques that map embedded applications onto multicore architectures. These techniques focus on optimizing performance, temperature distribution, reliability and fault tolerance for various models. Dongarra et al. [8] studied the problem of scheduling task graphs on a set of heterogeneous resources to maximize reliability and throughput, and proposed a throughput/reliability tradeoff strategy. Wang et al. [19] proposed replication-based scheduling for maximizing system reliability. The proposed algorithm incorporates task communication into system reliability and maximizes communication reliability by searching all optimal reliability communication paths for current tasks. These works explore the reliability of heterogeneous multicore processors from various aspects, and present efficient reliability improvement schemes, however, these works do not investigate the energy consumed by MPSoCs, which interplays with system reliability.

Extensive research effort has been devoted to reduce energy consumption of DVFS-enabled heterogeneous multi-core platforms considering system reliability. Zhang et al. [21] proposed a novel genetic algorithm based approach to improve system reliability in addition to energy savings for scheduling workflows in heterogeneous multicore systems. In [16], Spasic et al. presented a novel polynomial-time energy minimization mapping approach for synchronous dataflow graphs. They used task replication to achieve load-balancing on homogeneous processors, which enables processors to run at a lower frequency and consume less energy. In [7], Das et al. proposed a genetic algorithm to improve the reliability of DVFS-based MPSoC platforms while fulfilling the energy budget and the performance constraint. However, their task mapping approach tries to minimize core aging together with the

susceptibility to transient errors. In [11], the authors considered the problem of achieving a given reliability target for a set of periodic real-time tasks running on a multicore system with minimum energy consumption. The proposed framework explicitly takes into account the coverage factor of the fault detection techniques and the negative impact of DVFS on the rate of transient faults leading to soft errors. Although above works explore various techniques to save energy, to the best of our knowledge, none of the above works considers system throughput in addition to reliability and energy. Our approach is thus the first attempt to model both reliability, performance and energy for workflow scheduling in MPSoC.

3 Model

We consider the problem of scheduling a pipelined workflow onto a homogeneous multi-core platform that is subject to failures. The goal is to minimize the expected energy consumption for executing a single dataset, given some constraints on the expected and worst-case throughput of the workflow. In the following subsections, we detail how to model applications, platforms, failures, energy cost, period (which is the inverse of the throughput), and how to formally define the optimization problem.

3.1 Applications

We focus on linear chain workflow applications, where task dependencies form a linear chain: each task requests an input from the previous task, and delivers an output to the next task. There are n tasks $T_1, \ldots, T_n$. Furthermore, the application is pipelined, i.e., datasets continuously enter through the first task, and several datasets can be processed concurrently by the different tasks. Such applications are ubiquitous in processing of streaming datasets in the context of embedded systems [13].

We assume that the initial data resides in memory, and the final data stays in memory. Task T_j is characterized by a workload w_j, and the size of its output file to the next task o_j, as illustrated in Fig. 1 (except for the last task). In the example, we have $w_1 = 2$, $w_2 = 5$, $w_3 = 4$, and $o_1 = 3$, $o_2 = 1$. Once the first dataset reaches task T_3, while it is processed by T_3, dataset 2 is transferred between T_2 and T_3, dataset 3 is processed by T_2, dataset 4 is transferred between T_1 and T_2, and dataset 5 is processed by T_2. At the next period, all dataset numbers are incremented by one.

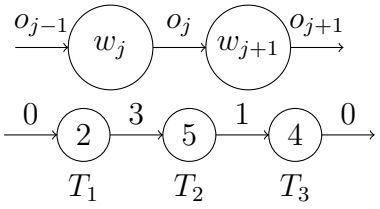


Figure 1: Linear chain workflow application.

3.2 Platforms

The target platforms are embedded systems composed of p homogeneous computing cores. Each core can run at a different speed with a corresponding error rate and an energy consumption. If task T_j is executed on a core operating at speed $s(j)$ and if it is not subject to a failure, it takes a time $\frac{w_j}{s(j)}$ to execute a single dataset.

We focus on the most widely used speed model, the *discrete* model, where cores have a discrete number of predefined speeds, which correspond to different voltages at which the core can be operating. Switching is not allowed during the execution of a given task, but two different executions of a task can be executed at different speeds. The set of speeds is $\{s_{\min} = s_1, s_2, \ldots, s_K = s_{\max}\}$. The *continuous* model is used mainly for theoretical studies, and let the speed take any value between the minimum speed $s_{\min}$ and the maximum speed $s_{\max}$.

All cores are fully interconnected by a network on chip (NoC). The bandwidth β is the same between any two cores, hence it takes $\frac{o_j}{\beta}$ for task T_j to communicate a dataset to task T_{j+1}. The network on chip enables cores to communicate simultaneously with others while they are computing, i.e., communications and computations can be overlapped. Therefore, while task T_j is processing dataset k, it is receiving the input for dataset $k-1$ from the previous task, and sending the output for dataset $k+1$ to the next task. Hence, these operations overlap, and take respectively a time $\frac{o_{j-1}}{\beta}$ and $\frac{o_j}{\beta}$.

We follow the model of [12, 5], where cores are equipped with a router, and on which there are registers. We can use the registers to store intermediate datasets, hence having *buffers* between cores. If datasets are already stored in the input buffer of a core, and if there is empty space in the output buffer, then the core can process a dataset without having to wait for the previous or next core.

3.3 Failure model and duplication

Embedded system platforms are subject to failures, and in particular transient errors caused by radiation. When subject to such errors, the system can return to a safe state and repeat the computation afterwards. According to the work of [25], radiation-induced transient failures follow a Poisson distribution. The fault rate is given by:

$$\lambda(s) = \lambda_0 e^{d\frac{s_{\max}-s}{s_{\max}-s_{\min}}},$$

where $s \in [s_{\min}, s_{\max}]$ denotes the running speed, d is a constant that indicates the sensitivity to dynamic voltage and frequency scaling, and λ_0 is the average failure rate at speed $s_{\max}$. λ_0 is usually very small, of the order of 10^{-5} per hour [2]. Therefore, we can assume that there are no failures when running at speed $s_{\max}$. We can see that a very small decrease of speed leads to an exponential increase of failure rate.

The failure probability of executing task T_j (without duplication) on a processor running at speed s_k is therefore $f_j(s_k) = \lambda(s_k)\frac{w_j}{s_k}$. If an error strikes, we resume the execution by reading the dataset again from local memory (i.e., the input has been copied before executing the task, we re-execute the task on the copy), and this re-execution is done at maximum speed so that no further error will strike the same dataset on this task. We assume that the time to prepare re-execution is negligible. Still, this slows down the whole workflow since other tasks may need to wait.

We propose to duplicate some tasks to mitigate the effect of failures and have a reliable execution. This means that two identical copies of a same task are executed on two distinct cores, both core running at the same speed. In this case, if a failure occurs in only one copy, we can keep going with the successful copy. However, it may increase the energy cost and communication cost. Similarly to one execution at the maximum speed, we assume that an error on a duplicated task is very unlikely (i.e., at least one copy will be successful), and hence $f_j(s_k) = 0$ if T_j is duplicated.

Let $m_j = 1$ if task T_j is duplicated, and $m_j = 0$ otherwise. Let s_k be the speed at which T_j is processed. The failure probability for T_j is therefore $f_j(s_k) = (1-m_j)\lambda(s_k)\frac{w_j}{s_k}$, i.e., it is zero if the task is duplicated, and $\lambda(s_k)\frac{w_j}{s_k}$ otherwise (the instantaneous error rate at speed s_k times the time to execute task T_j).

If we do not account for communications, the expected execution time of task T_j running at speed s_k is:

$$t_j = \frac{w_j}{s_k} + f_j(s_k)\frac{w_j}{s_{\max}}.$$

Indeed, with duplication, at least one execution will be successful, while with

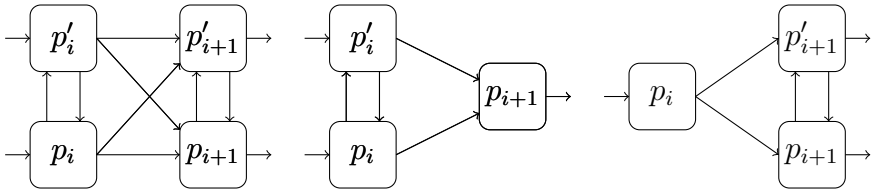


Figure 2: Communications with task duplication.

a single execution, if there is a failure, we re-execute the task at maximum speed and there are no further failures.

If a task is duplicated, this implies that further communications may be done, but they will occur in parallel. If T_j is duplicated, both processors p_j and p'_j on which T_j is executed are synchronized, and only one of them obtaining a correct result will do the output communication (to one or two processors, depending on whether T_{j+1} is duplicated or not), see Fig. 2 for different possible configurations. The synchronization cost is assumed to be negligible.

3.4 Energy

We follow a classical energy model, see for instance [3], where the dissipated power for running at speed s_k is s_k^3, and hence the energy consumed for a single execution of task T_j running at speed s_k is $\frac{w_j}{s_k} \times s_k^3 = w_j s_k^2$. We further account for possible failures and duplication, hence obtaining the expected energy consumption for T_j running at speed s_k for one dataset:

$$E_j(s_k) = (m_j + 1)w_j s_k^2 + f_j(s_k)w_j s_{\max}^2.$$

Indeed, if task T_j is duplicated ($m_j = 1$), we always pay for two executions ($2w_j s_k^2$) but there is no energy consumed following a failure, while without duplication ($m_j = 0$), we account for the energy consumed by the re-execution in case of a failure.

We assume that the energy consumed by communications and buffers is negligible compared to the energy consumed by computations, see [12]. Therefore, the expected energy consumption of the whole workflow to compute a single dataset is the sum of the expected energy consumption of all tasks.

3.5 Period definition and constraints

As stated before, each task is mapped onto a different processor, or a pair of processors when duplicated, and different tasks are processing different datasets. In steady-state mode, the throughput is either constrained by the

task with the longest execution time, or by the longest communication time, which is slowing down the whole workflow. The time required between the execution of two consecutive datasets corresponds to this bottleneck time and is called the period. It is the inverse of the throughput.

In this work, we are given a target period P_{t}, hence the target throughput is $\frac{1}{P_{\text{t}}}$. This corresponds for instance to the rate at which datasets are produced. We consider two different constraints: i) ensure that the expected period is not exceeding P_{t}, hence the target becomes a bound, and/or ii) ensure that the probability of exceeding the target P_{t} for a given dataset is not greater than $proba_{\text{t}}$. This second constraint corresponds to real-time systems, where a dataset is lost if its execution exceeds the target period P_{t}, and the probability $proba_{\text{t}}$ ($0 \leq proba_{\text{t}} \leq 1$) expresses how many losses are tolerated. If $proba_{\text{t}} = 1$, there is no constraint, while $proba_{\text{t}} = 0$ means that no losses are tolerated.

Recall that the objective is to minimize the expected energy consumption per dataset of the whole workflow. Some tasks may be duplicated, and each task may run at a different speed. The communication between two consecutive tasks T_j and T_{j+1} takes a constant time $\frac{o_j}{\beta}$, and it must fit within the target period. Therefore, we assume that for all $1 \leq j < n$, $\frac{o_j}{\beta} \leq P_{\text{t}}$.

We assume in this section that the set of duplicated tasks is know: we set m_i to 0 or 1 for each task T_i. Furthermore, let $s(i)$ be the speed at which task T_i is executed, for $1 \leq i \leq n$.

We first consider the case without failures and express the period in this case (Section 3.5.1). Then, we express the expected period when the platform is subject to failures in Section 3.5.2. Note that we assume that there is a sufficient number of buffers between cores, so that a failure does not necessarily impact the period, given that the cores have access to datasets stored in buffers, and can use empty buffers to store output datasets. Finally, we explain in Section 3.5.3 how to compute the probability that a dataset exceeds the target period P_{t}.

3.5.1 Period without failures

In the case without failures, the period is determined by the bottleneck task computation or communication: $P_{\text{nf}} = \max_{1 \leq i \leq n} \left\{ \frac{w_i}{s(i)}, \frac{o_i}{\beta} \right\}$.

We denote by L the set of tasks whose execution time is equal to P_{nf}, i.e., $L = \left\{ T_i | \frac{w_i}{s(i)} = P_{\text{nf}} \right\}$. If the bottleneck time P_{nf} is achieved by a communication, this set may be empty.

3.5.2 Expected period

We consider that each processor is equipped with three or more buffers, two of them holding an input (resp. output) dataset being received (resp. sent), and the other buffers are used for storing intermediate datasets: a buffer is filled when the task is completed, but the following processor is not yet ready to receive the next dataset (i.e., the output buffer is still in use). We consider the period in steady-state, after the initialization has been done, i.e., all processors are currently working on some datasets.

The set of tasks L is empty if the computation time for all tasks is strictly smaller than the period P_{nf}. When subject to a failure, tasks not in L can use data stored in buffers and process datasets at a faster pace than the period, until they have caught up with the time lost due to the failure.

However, errors in tasks of L are impacting the period, and therefore, if such a task is subject to failure, the re-execution time is added to the period. The expected period can therefore be expressed as follows:

$$P_{\text{exp}} = P_{\text{nf}} + \sum_{j \in L} f_j(s(j)) \frac{w_j}{s_{\max}}. \tag{1}$$

Indeed, the period P_{nf} is achieved when there is either no failure, or a failure in a task not in L. In case of a failure while executing a task T_j in L, the period is $P_{\text{nf}} + \frac{w_j}{s_{\max}}$, and this happens with a probability $f_j(s(j))$. As discussed before, we assume that there is no failure during re-execution, and that the probability of having two failures while executing a single dataset is negligible.

Note that this formula also holds when some tasks are duplicated. If task $j \in L$ is duplicated ($m_j = 1$), it will never fail and hence its period will be P_{nf}. In this case, $f_j(s(j)) = 0$ by definition, hence the formula remains correct.

3.5.3 Bounding the probability of exceeding the period bound

For the second constraint, we focus on the actual period of each dataset, rather than the expected period, and we estimate the probability at which the period of a dataset exceeds P_{t}. We consider that $P_{\text{nf}} \leq P_{\text{t}}$, otherwise the bound can never be reached, and the probability is always one.

The actual period, denoted by P_{act}, is a random variable that ranges from P_{nf} to $\max_{1 \leq i \leq n} \left(\frac{w_i}{s(i)} + \frac{w_i}{s_{\max}} \right)$. We define the set of tasks that may exceed the target period P_{t} in case of failures:

$$S_{\text{excess}} = \left\{ T_i \mid \frac{w_i}{s(i)} + \frac{w_i}{s_{\max}} > P_{\text{t}} \right\}.$$

Therefore, if a failure strikes a task in S_{excess} on a given dataset, the target period P_{t} may not be met for this dataset. An error happens on task i with probability $f_i(s(i))$. Since failures are independent, the period of a dataset will not exceed the bound if and only if no task in the set S_{excess} has a failure, i.e., this happens with a probability $\prod_{T_i \in S_{\text{excess}}}(1 - f_i(s(i)))$.

Hence, the probability of exceeding the bound is given by:

$$P(P_{\text{act}} > P_{\text{t}}) = 1 - \prod_{T_i \in S_{\text{excess}}} (1 - f_i(s(i))) \approx \sum_{T_i \in S_{\text{excess}}} f_i(s(i)), \qquad (2)$$

considering that the failure probabilities are small, and that $f_i(s(i)) \times f_j(s(j)) = 0$ for any $1 \leq i, j \leq n$. This approximation is in line with the assumption that we do not consider two consecutive failures in a same task.

Finally, the second constraint that we consider, after the one on the expected period described above, is to bound the probability of exceeding the target period P_{t} by the target probability $proba_{\text{t}}$:

$$P(P_{\text{act}} > P_{\text{t}}) \leq proba_{\text{t}}.$$

3.6 Optimization problems

The objective is to minimize the expected energy consumption per dataset of the whole workflow, and we consider two constraints. The goal is to decide which tasks to duplicate, and at which speed to operate each task. More formally, the problem is defined as follows:

(MinEnergy). *Given a linear chain composed of n tasks, a computing platform with p homogeneous cores that can be operated with a speed within set S, a failure rate function f, and a target period P_t, the goal is to decide, for each task T_j, whether to duplicate it or not (set $m_j = 0$ or $m_j = 1$), and at which speed to operate it (choose $s(j) \in S$), so that the total expected energy consumption is minimized, under the following constraints:*
i) The expected period P_{exp} should not exceed P_t;
ii) The probability of exceeding the target period P_t should not exceed the target probability $proba_t$.

Note that if there is a task k such that $P_{\text{t}} < \frac{w_k}{s_{\max}}$ or $P_{\text{t}} < \frac{o_k}{\beta}$, then there is no solution since the target period can never be met.

If P_{t} is large enough, the problem will not be constrained since in all solutions, the expected period will always be under the target period. This is the case for $P_{\text{t}} \geq \max(\frac{w_i}{s_{\min}} + \frac{w_i}{s_{\max}})$ and $P_{\text{t}} \geq \max(\frac{o_k}{\beta})$. In this case, each task running at the slowest possible speed, and being re-executed after a failure, will not exceed P_{t}. This problem without constraints is denoted as MinEnergy-NoC.

We also consider the particular cases where only one or the other constraint matters. MinEnergy-PerC is the problem where we do only consider the first constraint on the expected period (i.e., set $proba_{\mathrm{t}} = 1$), while MinEnergy-ProbaC is the problem where we do only focus on the probability of exceeding the target period, i.e., we do not consider P_{exp}.

4 Complexity analysis

4.1 Without errors

When the workflow is free of errors, a task T_j running at speed s_j takes exactly a time $\frac{w_j}{s_j}$, and consumes an energy of $w_j s_j^2$. Hence, to minimize the energy consumption, one must use the smallest possible speed such that the target period is not exceeded, hence $s_j = \max\left\{\frac{w_j}{P_{\mathrm{t}}}, s_{\min}\right\}$ in the continuous case. Since we consider discrete speeds, the optimal speed for task T_j is therefore the smallest speed larger than or equal to $\frac{w_j}{P_{\mathrm{t}}}$ within the set of possible speeds. This is true for all tasks, hence the problem can be solved in polynomial time.

4.2 Without constraints

We consider the MinEnergy-NoC problem, and propose the **BestEnergy** algorithm to optimally solve this problem. The idea is to use the speed that minimizes the energy consumption for each task, since we do not have any constraint about exceeding the target period. For each task, either we execute it at this optimal speed, or it may be even better (in terms of energy consumption) to duplicate it and run it at the smallest possible speed $s_{\min}$.

Theorem 1. MinEnergy-NoC *can be solved in polynomial time, using the* ***BestEnergy*** *algorithm, both for the discrete and for the continuous energy model.*

Proof. Given a task of weight w executed at speed s without duplication, the energy consumption is $E(s) = w \times s^2 + \lambda_0 w^2 s_{\max}^2 \frac{e^{d\frac{s_{\max}-s}{s_{\max}-s_{\min}}}}{s}$. The (continuous) speed that minimizes this energy consumption can be obtained by deriving $E(s)$:

$$E'(s) = 2ws - \left(\frac{\lambda_0 w^2 s_{\max}^2}{s^2} + \frac{\lambda_0 d w^2 s_{\max}^2}{s(s_{\max} - s_{\min})}\right) e^{d\frac{s_{\max}-s}{s_{\max}-s_{\min}}}.$$

$E'(s)$ is a monotonically increasing function, and we let s^* be the speed such that $E'(s^*) = 0$, hence $E(s^*)$ is minimum.

If the task is not duplicated, for MinEnergy-NoC-Cont, the optimal speed is $s_{\text{opt}} = \max\{s^*, s_{\min}\}$. In the discrete case MinEnergy-NoC-Disc, s_{opt} is simply the speed that minimizes the energy consumption, hence $s_{\text{opt}} = \text{argmin}_{s\in\{s_{\min},\ldots,s_{\max}\}}\{E(s)\}$.

Now, if the task is duplicated, we assume that it will not be subject to error, hence the energy consumption at speed s is $2ws^2$. Therefore, it is minimum when the task is executed at the minimum speed, and the corresponding energy consumption is $2ws_{\min}^2$ (both in the discrete and continuous case).

BestEnergy is a greedy algorithm that sets the speed of each task at s_{opt} (not using duplication), and then greedily assigns remaining processors to tasks that would gain most from being duplicated (if any), see Algorithm 1. It is easy to see that it is optimal, since any other solutions could only have a greater energy consumption. □

Algorithm 1 – **BestEnergy**(n, p)

for $i = 1$ to n **do**
  Compute $s_{\text{opt}}(i)$ for task T_i, the speed that minimizes energy consumption if T_i is not duplicated;
  $s_i \leftarrow s_{\text{opt}}(i)$, $m_i \leftarrow 0$;
  $g_i \leftarrow E(s_i) - 2w_i s_{\min}^2$ (Possible gain in energy if T_i is duplicated);
end for
Sort tasks by non-increasing g_i, T_j is the task with max g_i;
$p_{av} \leftarrow p - n$ (Number of available processors);
while $g_j > 0$ and $p_{av} > 0$ **do**
  $m_j \leftarrow 1$, $s_j \leftarrow s_{\min}$, $p_{av} \leftarrow p_{av} - 1$;
  $j \leftarrow$ the index of next task in the sorted list;
end while
return $< s_i, m_i >$;

4.3 With the probability constraint

We now prove that the decision version of MinEnergy-ProbaC is NP-complete. In the decision version of MinE-Dec, the goal is to find an assignment set of speeds such that the probability of exceeding the target period P_{t} does not exceed $proba_{\text{t}}$, and such that the energy consumption does not exceed a given energy threshold E_{t}. The proof is based on a reduction from the Partition problem, known to be NP-complete [10]. The idea is to have only two possible speeds, and one must decide at which speed to

operate each task. We set as many processors as tasks, so that no duplication can be done.

Theorem 2. MinE-Dec *is NP-complete, even when duplicating tasks is not possible.*

Proof. We first check that MinE-Dec is in NP: given a speed for each task, it is easy to verify in polynomial time whether the bounds on the failure probability and on the energy consumption are satisfied.

The proof of completeness is based on a reduction from the Partition problem, known to be NP-complete [10]. We consider an instance $\mathcal{I}_{1-\mathrm{Par}}$ of 2-partition: given n positive integers $a_1, \ldots, a_n$, does there exist a partition of $\{1, \ldots, n\}$ into two subsets I_1 and I_2 ($I_1 \cup I_2 = \{1, \ldots, n\}$ and $I_1 \cap I_2 = \emptyset$) such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i = S/2$, where $S = \sum_{i=1}^{n} a_i$?

We let $\Delta = \max a_j / \min a_j$. We build an instance $\mathcal{I}_{2-\mathrm{MinE}}$ of MinE-Dec as follows:

- The workflow is made of n tasks, of size $w_1 = a_1, \ldots, w_n = a_n$, to be processed on $p = n$ cores (no duplication is possible).
- There are only two possible speeds, $s_{\min} = s_1 = 1$, and $s_{\max} = s_2 = 2\Delta$.
- The failure rate function for these speeds is given by $f(s_1) = 1/S$ and $f(s_2) = 0$.
- We set the target period to $P_{\mathrm{t}} = \min_i w_i$, the bound on the probability of exceeding P_{t} to $proba_{\mathrm{t}} = 1/2$, and the bound on the energy to $E_{\mathrm{t}} = 2\Delta^2(S+1) + S/2$.

We first assume that there is a solution (I_1, I_2) to instance $\mathcal{I}_{1-\mathrm{Par}}$. For the MinE-Dec problem, we set all tasks T_i with $i \in I_1$ to speed $s_1 = 1$, and all tasks T_i with $i \in I_2$ to speed $s_2 = 2\Delta$. Given the target period, we check that $S_{\mathrm{excess}} = I_1$:

- For any task in I_1, we have $w_i/s_1 + w_i/s_{\max} > w_i \geq \min w_j = P_{\mathrm{t}}$.
- For any task in I_2, we have $w_i/s_2 + w_i/s_{\max} = 2w_i/(2\Delta) = w_i \min w_j / \max w_i \leq \min w_j = P_{\mathrm{t}}$.

Thus, the probability of exceeding P_{t} is given by

$$\sum_{i \in I_1} f(s_1) w_i / s_1 = 1/S \sum_{i \in I_1} w_i = 1/S \times S/2 = 1/2 = proba_{\mathrm{t}},$$

which satisfies the constraint on the probability. Then, we compute the energy of the obtained solution:

$$\begin{aligned} E &= \sum_{i\in I_1}(w_i s_1^2 + f(s_1) w_i s_{\max}^2) + \sum_{i\in I_2} w_i s_2^2 \\ &= S/2(1 + 4\Delta^2/S) + S/2 \cdot 4\Delta^2 = E_{\mathrm{t}}, \end{aligned}$$

which satisfies the bound on the energy. Hence, we have found a solution to $\mathcal{I}_{2-\mathrm{MinE}}$.

We then assume that $\mathcal{I}_{2-\mathrm{MinE}}$ has a solution. We denote by I_1 the set of tasks running at speed s_1, and by I_2 the others, running at speed s_2. As outlined below, only tasks in I_1 contribute to the probability of exceeding P_{t}, and its bound writes:

$$\sum_{i\in I_1} f(s_1) w_i / s_1 \leq 1/2$$

With $s_1 = 1$ and $f(s_1) = 1/S$, this gives $\sum_{i\in I_1} w_i \leq S/2$. The bound on the energy writes:

$$\begin{aligned} \sum_{i\in I_1}(w_i s_1^2 + f(s_1) w_i s_{\max}^2) + \sum_{i\in I_2} w_i s_2^2 &\leq 2\Delta^2(S+1) + S/2 \\ \sum_i w_i s_1^2 + \sum_{i\in I_1} w_i/S s_{\max}^2 + \sum_{i\in I_2} w_i(s_2^2 - s_1^2) &\leq 2\Delta^2(S+1) + S/2 \\ S + \sum_{i\in I_1} w_i/S s_{\max}^2 + \sum_{i\in I_2} w_i(4\Delta^2 - 1) &\leq 2\Delta^2(S+1) + S/2 \end{aligned}$$

$$\begin{aligned} \sum_{i\in I_2} w_i(4\Delta^2 - 1) &\leq 2\Delta^2(S+1) - S/2 - \sum_{i\in I_1} w_i/S s_{\max}^2 \\ \sum_{i\in I_2} w_i(4\Delta^2 - 1) &\leq 2\Delta^2(S+1) - S/2 \\ \sum_{i\in I_2} w_i(4\Delta^2 - 1) &\leq S/2(4\Delta^2 - 1) + 2\Delta^2 \\ \sum_{i\in I_2} w_i &\leq S/2 + \frac{2\Delta^2}{4\Delta^2 - 1} \end{aligned}$$

Since $2\Delta^2/(4\Delta^2 - 1) < 1$ as soon as $\Delta \geq 1$ and all w_i's are integers, this gives $\sum_{i\in I_2} w_i \leq S/2$. Together with $\sum_{i\in I_1} \leq S/2$, this proves that I_1, I_2 is a solution to $\mathcal{I}_{1-\mathrm{Par}}$, which concludes the proof. □

5 Heuristics

We provide here several heuristics, all designed for the more realistic case of discrete speeds. Heuristics adapted for the case of continuous speeds are presented in Appendix A. We start with basic heuristics that will be used as baseline. Then we design heuristics aiming at bounding the expected period, and finally heuristics for bounding the probability of exceeding the target period.

5.1 Baseline heuristics

We first outline the baseline heuristics that will serve as a comparison point, but may not satisfy the constraints. First, the **BestEnergy** algorithm described in Section 4.2 is providing a lower bound on the energy consumption, but since it means that many tasks are running at the minimum speed, we expect the period to be large, and it may well exceed the bound.

Another simple solution consists in having each task executed at the maximum speed $s_{\max}$. We refer to this heuristic as **MaxSpeed** (see Algorithm 2).

Algorithm 2 – **MaxSpeed**(n, p)

for $i = 1$ to n **do**
  $s_i \leftarrow s_{\max}$, $m_i \leftarrow 0$;
end for
return $< s_i, m_i >$;

The third baseline heuristic, **DuplicateAll** (see Algorithm 3), duplicates all tasks, assuming that there are twice more processors than tasks ($p \geq 2n$), and the corresponding speeds for each tasks used in this case are the ones

Algorithm 3 – **DuplicateAll**(n, p)

if $p \geq 2n$ **then**
  for $i = 1$ to n **do**
    choose s_i as the smallest possible speed so that $\frac{w_i}{s_i} \leq P_{\mathrm{t}}$;
    $m_i \leftarrow 1$;
  end for
  return $< s_i, m_i >$;
else
  return *failure*;
end if

derived in Section 4.1. Indeed, there will not be any errors in this case, and we aim at respecting the target period P_{t}.

Note that both **MaxSpeed** and **DuplicateAll** will always satisfy the bounds, since there will be no errors, and hence the expected period is equal to the period without failure. However, both heuristics may lead to a large waste of energy. They provide an upper bound on the energy consumption when using a naive approach.

5.2 Bounding the expected period

In this section, we focus on the constraint on the expected period, hence targeting the MinEnergy-PerC problem.

5.2.1 Heuristic Threshold

The **Threshold** heuristic aims at reaching the target expected period P_{t} (see Algorithm 4). The first step consists in setting all task speeds to the smallest speed such that $\frac{w_i}{s_i} \leq P_{\mathrm{t}}$. If P_{exp} is still larger than P_{t}, then one of the tasks with largest duration ($\frac{w_i}{s_i}$) is duplicated: this allows P_{nf} to be smaller than P_{t} and constant from this moment on. Note that in the special case of a communication time reaching P_{t}, there is no need to duplicate a task to have $P_{\mathrm{nf}} = P_{\mathrm{t}}$.

From Equation (1), $P_{\mathrm{exp}} = P_{\mathrm{nf}} + \sum_{j \in L} f_j(s(j)) \frac{w_j}{s_{\max}}$. We made sure that P_{nf} is smaller than or equal to P_{t}. In order to make $P_{\mathrm{exp}} \leq P_{\mathrm{t}}$, each task T_i of L has to be either run at a higher speed (which removes it from L), or duplicated (which sets $f_{(s(i))}$ to 0). We greedily duplicate tasks for which duplication costs less energy, until there remains no more processors. Then, we speed up other tasks. Finally, we use the same technique as in **BestEnergy** to attempt to reduce again the energy of non-duplicated tasks: if the minimum speed s for energy consumption is larger than the actual speed s_i of a task T_i, its speed is increased to s.

Algorithm 4 – **Threshold**(n, p)

for all tasks T_i **do**
  $s_i \leftarrow$ the smallest speed such that $\frac{w_i}{s_i} \leq P_{\mathrm{t}}$, $m_i \leftarrow 0$;
end for
$p_{av} \leftarrow p - n$ (number of available processors);
if $P_{\mathrm{t}} > \max(\frac{o_k}{\beta})$ and $p_{av} \geq 1$ **then**
  Select a task T_k with largest duration (break tie by selecting one with smallest w_k), set $m_k \leftarrow 1$ and $p_{av} \leftarrow p_{av} - 1$;
end if
if $P_{\mathrm{exp}} > P_{\mathrm{t}}$ **then**
  $Q \leftarrow \{$tasks of L with $m_i = 0\}$;
  for all tasks T_j in Q **do**
    $s \leftarrow$ the smallest speed that is larger than s_j;
    $g_j \leftarrow w_j * (s^2 + f_j(s)s_{max}^2 - 2s_j^2)$ (Possible gain in energy if T_j is duplicated);
  end for
  Sort tasks of Q by non-increasing g_i;
  for all task T_j in Q **do**
    if $p_{av} > 0$ **then**
      $m_j \leftarrow 1$, $p_{av} \leftarrow p_{av} - 1$;
    else
      $s_j \leftarrow$ the smallest speed that is larger than s_j;
    end if
  end for
end if
for all task T_i with $m_i = 0$ **do**
  Compute the speed s that minimizes $E_i(s)$;
  if $s_i < s$ **then**
    $s_i \leftarrow s$;
  end if
end for
return $< s_i, m_i >$;

5.2.2 Heuristic Closer

The previous **Threshold** heuristic uses duplication: at least one task is duplicated (in order to fix P_{nf}), which requires spare processors. We propose another heuristic that does not have this requirement. In the **Closer** heuristic (see Algorithm 5), after setting all task speeds to the smallest ones so that $\frac{w_i}{s_i} \leq P_{\mathrm{t}}$, we increase the speed of all tasks in L while $P_{\mathrm{exp}} > P_{\mathrm{t}}$ by scaling

all tasks simultaneously: we set a coefficient and make sure that for each task, its speed is not smaller than $coef \times s'_i$, where s'_i is the initial speed of T_i. The coefficient is gradually increased until $P_{\text{exp}} \leq P_{\text{t}}$. Finally, we use the same technique as in **BestEnergy** to attempt to further reduce the energy consumption of tasks.

Algorithm 5 Closer$(n, p, \Delta s)$

for all tasks T_i **do**
  $s'_i \leftarrow$ the smallest speed such that $\frac{w_i}{s_i} \leq P_{\text{t}}$, $m_i \leftarrow 0$;
end for
Set $coef \leftarrow 1$;
while $P_{\text{exp}} > P_{\text{t}}$ **do**
  $coef \leftarrow coef + \Delta s$;
  for all tasks T_i of set L **do**
    $s_i \leftarrow$ the smallest speed that is not smaller than $coef \times s'_i$;
  end for
end while
for all task T_i **do**
  Compute the speed s that minimizes $E_i(s)$;
  if $s_i < s$ **then**
    $s_i \leftarrow s$;
  end if
end for
return $< s_i, m_i >$

5.3 Bounding the probability of exceeding P_{t}

In this section, we design a heuristic focusing on the constraint on the probability of exceeding P_{t}, thus for the MinEnergy-ProbaC problem.

BestTrade (see Algorithm 6) aims at finding the best tradeoff between energy consumption and the probability of exceeding P_{t}. We consider for each task two critical speeds:

- s_c^i is the speed such that $w_i/s_c^i + w_i/s_{\max} = P_{\text{t}}$; it corresponds to the minimum speed that a task can take without belonging to the S_{excess} set;
- $s_d^i = w_i/P_{\text{t}}$ is the minimum speed that can be assigned to a task: if it is set to a smaller speed, its duration will always exceed the target period.

The idea of the algorithm is first to set all tasks to the smallest speeds that are not smaller than their s_c^i speed. For some tasks, this might be equal to their minimum speed (the smallest possible speed not smaller than s_d^i). In this case, there is no room for reducing speed again without exceeding the target period. For other tasks, we sort them by non-increasing weight: tasks with higher weights contribute the most to the energy dissipation and are thus first slowed down: we select the task T_i with the largest weight, reduce its speed to the minimum possible speed not smaller than w_i/P_{t}. We continue with the tasks of smaller weight, until $P(P_{\mathrm{act}} > P_{\mathrm{t}}) \geq \mathit{proba}_{\mathrm{t}}$. At last, if $P(P_{\mathrm{act}} > P_{\mathrm{t}}) > \mathit{proba}_{\mathrm{t}}$, we just undo the last move to make $P(P_{\mathrm{act}} > P_{\mathrm{t}}) < \mathit{proba}_{\mathrm{t}}$.

We then consider duplication: if duplicating a task T_i (and setting its speed to the smallest speed that is not smaller than s_d^i) is beneficial compared to the current solution (and if a processor is available), the task is duplicated.

Algorithm 6 BestTrade(n, p)

We assume all weights are different ($w_i \neq w_j$ for $i \neq j$);
for $j = 1$ **to** $j = n$ **do**
  $s_j \leftarrow$ smallest possible speed not smaller than $w_j/(P_{\mathrm{t}} - w_j/s_{\max})$, $m_j = 0$;
end for
$S_{\mathit{reduce}} \leftarrow$ tasks that have possible speeds between w_j/P_{t} and $w_j/(P_{\mathrm{t}} - w_j/s_{\max})$;
sort tasks of S_{reduce} by non-increasing weight;
$k = 1$;
while $P(P_{\mathrm{act}} > P_{\mathrm{t}}) < \mathit{proba}_{\mathrm{t}}$ **do**
  Reduce speed of k-th task in S_{reduce} to the smallest that is not smaller than w_j/P_{t};
  $k = k + 1$;
end while
if $P(P_{\mathrm{act}} > P_{\mathrm{t}}) > \mathit{proba}_{\mathrm{t}}$ **then**
  set speed of (k-1)-th task in S_{reduce} to the smallest that is not smaller than $w_j/(P_{\mathrm{t}} - w_j/s_{\max})$;
end if
$p_{av} \leftarrow p - n$ (Number of available processors);
for $j = 1$ **to** $j = n$ **do**
  $s_d \leftarrow$ the smallest speed that is not smaller than w_j/P_{t};
  if $2w_j s_d^2 < w_j s_j^2 + f_j(s_j) w_j s_{\max}^2$ and $p_{av} > 0$ **then**
    duplicate $T_j, m_j = 1, s_j = s_d, p_{av} \leftarrow p_{av} - 1$;
  end if
end for

6 Experimental validation through simulations

In this section, we evaluate all proposed algorithms through extensive simulations on both real applications and synthetic ones, in the case of discrete speeds. Results with continuous speeds are available in Appendix B. For reproducibility purposes, the code is available at `https://github.com/gouchangjiang/Pipeline_on_MPSoC`.

Given a computing platform and an application, we set the target period P_{t} and probability $proba_{\mathrm{t}}$ so that all assumptions made in the model are true:

- When all tasks are executed with the minimum speed $s_{\min}$, the maximum failure rate is not larger than 10^{-2}. With such a failure rate, the failure of two copies of a duplicated task is very unlikely, and the approximation in Equation (2) holds.
- When all tasks are processed with speed $s_{\max}$, the maximum failure rate is not larger than 10^{-4}, which means that the failure of a task running at maximum speed is very unlikely.
- P_{t} should not be smaller than any task duration when running at maximum speed, otherwise, there is no way to meet the target period: $P_{\mathrm{t}} \geq \frac{\max(w_i)}{s_{\max}}$.

We set $P_{\mathrm{t}} = a + \kappa * (b - a)$, where $a = \max(w_i/s_{\max}, o_i)$ and $b = max(w_i/s_{\min} + w_i/s_{\max}, o_i)$: a (respectively b) is the maximal time spent on a task (either on computation or on communication), when running at the maximum (resp. minimum) speed. This way, P_{t} is never smaller than a, which satisfies the third condition above. Similarly, we avoid the case $P_{\mathrm{t}} \geq b$, in which the target is too loose, as even the minimum speed can achieve it. A small κ leads to a tighter target period. Under the above three conditions, we set κ to values from 0.05 to 0.95, by increment of 0.01. The target probability is set to $proba_{\mathrm{t}} = 0.05$ for synthetic applications and $proba_{\mathrm{t}} = 0.01$ for real applications.

We use the result of heuristic **BestEnergy** described in Section 4.2 as a comparison basis, as it gives the minimum energy consumption of the system without any constraint.

6.1 Multi-core embedded systems

We simulate a multi-core computing platform with 512 cores. Based on AsAP2 and KiloCore, two state-of-art MPSoCs described in Section 2, the frequency/voltage options are listed in Table 1. NoC on chips enables extremely fast communications. We describe the value of β together with the

output (input) file sizes o_i below in the next subsection. The failure rate is computed as described in Section 3.3 as $\lambda(s) = \lambda_0 e^{d\frac{s_{\max}-s}{s_{\max}-s_{\min}}}$. Based on the settings in [25], we set $\lambda_0 = 10^{-6}$ and $d = 4$.

Possible frequency/voltage	Normalized speed	Failure rate ($\times 10^{-6}/second$)
1.2 Ghz/1.3 V	1	1
987 Mhz/1.16 V	0.80	2.30
744 Mhz/1.03 V	0.61	5.29
502 Mhz/0.89 V	0.41	12.18
260 Mhz/0.75 V	0.21	28.01
66 Mhz/0.675 V	0.055	54.60

Table 1: Configurations of computing platforms.

6.2 Streaming applications

We use a benchmark proposed in [17] for testing the **StreamIt** compiler. It collects many applications from varied representative domains, such as video processing, audio processing and signal processing. The stream graphs in this benchmark are mostly parametrized, i.e., graphs with different lengths and shapes can be obtained by varying the parameters. Table 2 lists some linear chain applications (or application whose major part is a linear chain) from [17]. Some applications, such as time-delay equalization, are more computation intensive than others.

Following the same idea, we also generated synthetic applications in order to test the algorithms on larger applications. We generated 100 groups of linear chains. Each group contains 3,000 linear chains with the same number of nodes, which range from $0.01p$ to p from group to group by an increment $0.01p$, where p is the number of cores. The weights of the nodes w_i follow a truncated normal distribution with mean value 2,000, where the values smaller than 100 or larger than 4,000 are removed. The standard deviation is 500. This ensures that the execution time is not too long so that failure rate is acceptable. The communication time ($\frac{o_i}{\beta}$) follows a truncated normal distribution with mean value $0.001 * P_{\mathrm{t}}$, values that are larger than P_{t} are replaced by P_{t}. Here $P_{\mathrm{t}} = a + 0.05 * (b - a)$.

6.3 Simulation result

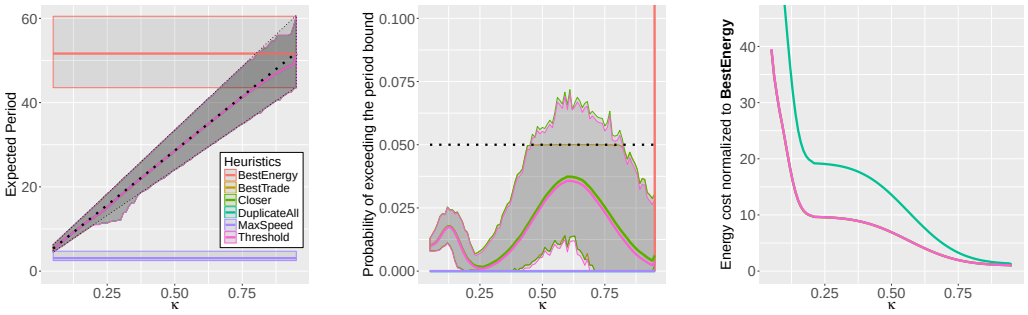
We present both results on synthetic applications and on real applications. On each plot, we show the minimum, mean, and maximum values of each

Application	Size	Average node's weight
CRC encoder	46	14.20
N-point FFT (coarse-grained)	13	1621.31
Frequency hopping radio	16	11815.81
16x oversampler	10	2157.4
Radix sort	13	179.92
Raytracer (rudimentary skeleton)	5	142.8
Time-delay equalization	27	23264.78
Insertion sort	6	475.83

Table 2: Real application examples.

heuristic. In some cases, only the mean is plotted to ease readability, when the minimum and maximum do not bring any meaningful information.

6.3.1 Synthetic applications

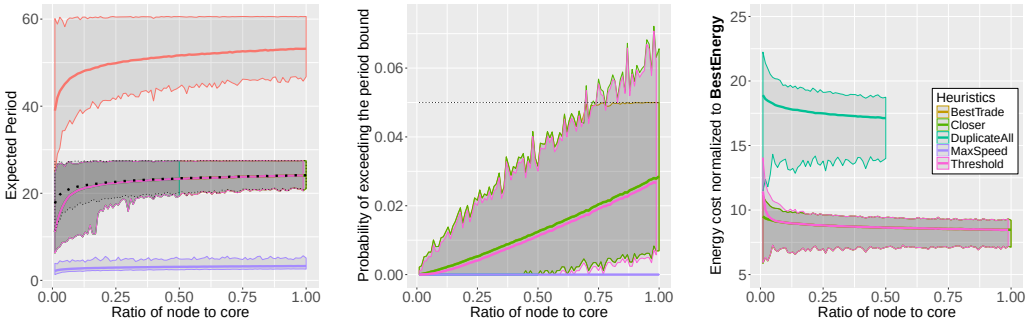


(a) Expected period (b) Probability (c) Energy consumption

Figure 3: Energy consumption and constraints on synthetic applications, as a function of κ.

Fig. 3 presents the results of all heuristics, both in terms of energy consumption, and in terms of constraints, when we vary the parameter κ, hence the tightness of the bound on the expected period. On Fig. 3a, the dashed lines represent the minimum, maximum and average period bound. All chains have $0.5p$ nodes. Apart from **MaxSpeed**, which always meets the bound, and **BestEnergy**, which never meets the bound, all heuristics succeed to meet the bound on the expected period. **BestTrade**, **Closer** and **DuplicateAll** are overlapped by **Threshold**.

Fig. 3b shows the probability of exceeding the period bound, and the dashed line is the target $proba_{\mathrm{t}}$. **DuplicateAll** is overlapped by **MaxSpeed**, and only **BestTrade** succeeds to always meet the bound. **Closer** and



(a) Expected period (b) Probability (c) Energy consumption

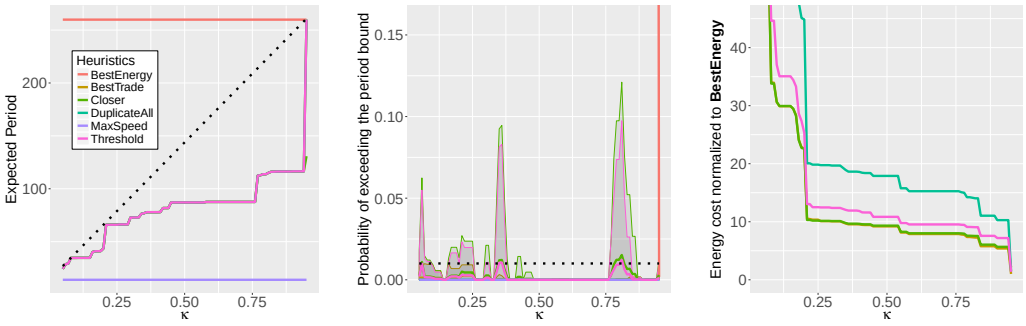
Figure 4: Energy consumption and constraints on synthetic applications, as a function of nodes to cores ratio.

Threshold give very similar results, but sometimes exceed the bound (when $0.4 \leq \kappa \leq 0.8$). **BestEnergy** may result in a probability of 0 when $\kappa = 0.95$, but for other values κ, its probability is always 1, which is not depicted in the figure.

Finally, Fig. 3c depicts the energy consumption, normalized by the result of **BestEnergy**. It does not include the energy cost of heuristic **MaxSpeed**, which is 215 times larger than **BestEnergy**. **Closer**, **Threshold** and **BestTrade** are very close to each other in this set of simulations, so some of them are overlapped. **DuplicateAll** consumes significantly more energy than the other heuristics. Overall, Fig. 3 shows that **BestTrade** is the best heuristic for these applications: it allows us to always meet both the expected period bound and the probability bound, and it offers similar energy performance as other heuristics. **Threshold** and **Closer** are also good options, however they often exceed the probability bound.

Fig. 4 illustrates performance of the heuristics for energy consumption and the two constraints, as a function of nodes to cores ratio. Given an amount of cores, a larger ratio corresponds to chains with more nodes. κ is set to 0.4 in these simulations. In Fig. 4a, the dashed lines represent the minimum, maximum and average period bound. On this figure, all heuristics except **BestEnergy** always meet the target period bound. **BestTrade**, **Closer**, **DuplicateAll** and **Theshold** overlap, except for their definition domain: **DuplicateAll** produces a valid allocation only for a ratio of nodes to cores smaller than or equal to 0.5, and **Threshold** requires to duplicate at least one node. Only **BestTrade** and **Closer** are defined for the whole range of the ratio.

Fig. 4b shows the probability of exceeding the period bound, where the dashed line is the target probability. Only **BestTrade** can meet the proba-



(a) Expected period (b) Probability (c) Energy consumption

Figure 5: Energy consumption and constraints on real applications, as a function of κ.

bility bound for all ratios. **Threshold** and **Closer** give very similar results as **BestTrade**, except on large ratios (i.e., large chains), where they exceed the bound. The probability for **DuplicateAll** and **MaxSpeed** is always zero.

Finally, Fig. 4c depicts the energy consumption as a function of the chain sizes. **MaxSpeed** is again too large to be included. The energy cost of **BestTrade** is the same as **Closer** and they are close to **Threshold**. As the size of the chains increases, the energy consumption of other heuristics get close to **BestEnergy**, and the difference between themselves also get smaller. Once again, Fig. 4 shows that **BestTrade** is the best option for all constraints.

6.3.2 Real applications

Fig. 5 shows the performance of the heuristics on real applications, as a function of κ. In Fig. 5a, the dashed line represents the average period bound. **BestTrade**, **DuplicateAll**, **Closer** and **Threshold** give very similar results and thus overlap. All heuristics except **BestEnergy** meet the period target. For probability bound, as shown in Fig. 5b, only **BestTrade** always meet the target. **DuplicateAll** and **MaxSpeed** both give a probability of 0 as before. **Closer** and **Threshold** sometimes exceed the target probability by a large factor. Finally Fig. 5c shows the energy required by each heuristics. In this setting, **BestTrade** is the most energy saving heuristic, closely followed by **Closer**. **Threshold** requires more energy, and **DuplicateAll** even more. Not surprisingly, **MaxSpeed** is the heuristic that costs the most energy, around 254 times larger than **BestEnergy** (so it is not included in Fig. 5c). This shows that **BestTrade** is the best heuristic also for real applications.

7 Conclusion

In this paper, we have studied the problem of optimizing the energy consumption of linear chain applications on MPSoCs, which have both reliability and performance constraints. We proposed a new model that allows us to change the frequency of the cores for different tasks and to duplicate some tasks. It takes into account both the expected period, the probability of exceeding the period and the energy efficiency. We proved that minimizing the energy consumption is easy without performance and reliability constraints, but that the problem becomes NP-complete when adding these constraints and when considering a discrete set of possible speeds. We then proposed several heuristics for choosing the tasks' processing speed and which tasks to duplicate. One of the proposed heuristics, **BestTrade**, is able to meet both bounds on the expected period and on the probability of exceeding the target period, while reducing the energy consumption.

Future work will target more complex allocation schemes, in which several tasks may be mapped on the same core, and more complex task graphs than linear chains (i.e., general directed acyclic graphs). Based on the present results, we expect the problems to become even more complex, but we believe that it will be possible to reuse some ideas derived from the study of linear chains.

References

[1] S. Albers and M. Hellwig. Online makespan minimization with parallel schedules. *Algorithmica*, 78:492–520, 2017.

[2] I. Assayad, A. Girault, and H. Kalla. Tradeoff exploration between reliability, power consumption, and execution time for embedded systems. *International Journal on Software Tools for Technology Transfer*, 15:229–245, 2013.

[3] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proc. of Int. Parallel and Distributed Processing Symp. (IPDPS)*, 2003.

[4] G. Blake, R. G. Dreslinski, and T. Mudge. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6):26–37, November 2009.

[5] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, and al. Kilocore: A fine-grained 1,000-processor array for task-parallel applications. *IEEE Micro*, 37:63–69, 2017.

[6] G. Chen, K. Huang, and A. Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination. *ACM Trans. on Embedded Computing Systems*, 13:111:1–111:21, 2014.

[7] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele. Combined dvfs and mapping exploration for lifetime and soft-error susceptibility improvement in mpsocs. In *Proc. of Design, Automation & Test in Europe (DATE)*, pages 61:1–61:6, 2014.

[8] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 280–288, 2007.

[9] G. P. Fettweis. The tactile internet: Applications and challenges. *IEEE Vehicular Technology Magazine*, 9:64–70, 2014.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, USA, 1990.

[11] M. A. Haque, H. Aydin, and D. Zhu. On reliability management of energy-aware real-time systems through task replication. *IEEE Trans. on Parallel and Distributed Systems*, 28(3):813–825, 2017.

[12] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):551–562, 2005.

[13] K. Huang, W. Haid, L. Bacivarov, M. Keller, and L. Thiele. Embedding formal performance analysis into the design cycle of mpsocs for real-time streaming applications. *ACM Trans. on Embedded Computing Systems*, 11:8:1–8:23, 2012.

[14] K. Li, X. Tang, and K. Li. Energy-efficient stochastic task scheduling on heterogeneous computing systems. *IEEE Trans. on Parallel and Distributed Systems*, 25(11):2867–2876, 2014.

[15] P. Marwedel, J. Teich, G. Kouveli, L. Bacivarov, L. Thiele, and et al. Mapping of applications to mpsocs. In *Proc. of Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 109–118, 2011.

[16] J. Spasic, D. Liu, and T. Stefanov. Energy-efficient mapping of real-time applications on heterogeneous mpsocs using task replication. In *Proc. of Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct 2016.

[17] W. Thies. *Language and Compiler Support for Stream Programs*. PhD thesis, MIT, Cambridge, MA, USA, 2009.

[18] D. Truong, W. Cheng, T. Mohsenin, and et al. A 167-processor 65 nm computational platform with per-processor dynamic supply voltage and dynamic clock frequency scaling. In *IEEE Symposium on VLSI Circuits*, pages 22–23, 2008.

[19] S. Wang, K. Li, J. Mei, G. Xiao, and K. Li. A reliability-aware task scheduling algorithm based on replication on heterogeneous computing systems. *Journal of Grid Computing*, 15(1):23–39, Mar 2017.

[20] W. Wolf, A. A. Jerraya, and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, 2008.

[21] L. Zhang, K. Li, C. Li, and K. Li. Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. *Information Sciences*, 379:241–256, 2017.

[22] S. Zhang, J. Wu, and S. Lu. Distributed workload dissemination for makespan minimization in disruption tolerant networks. *IEEE Transactions on Mobile Computing*, 15:1661–1673, 2016.

[23] B. Zhao, H. Aydin, and D. Zhu. Generalized reliability-oriented energy management for real-time embedded applications. In *Proc. of Design Automation Conference (DAC)*, pages 381–386, 2011.

[24] W. Zheng and R. Sakellariou. Stochastic dag scheduling using a monte carlo approach. *Journal of Parallel and Distributed Computing*, 73(12):1673 – 1689, 2013.

[25] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. *ACM Trans. on Embedded Computing Systems*, 10:26:1–26:27, 2011.

A Continuous-speed heuristics

A.1 Heuristic ThresholdC for continuous speeds

Heuristic **ThresholdC** is based on the same ideas but designed for the case when continuous speeds are available. In the first step, tasks speeds are initialized to the speeds which makes $\frac{w_i}{s_i} = P_{\mathrm{t}}$. Then duplicate a specific task to make $P_{\mathrm{nf}} = P_{\mathrm{t}}$. After it, we speed up tasks in L or duplicate them. We proceed similarly as in **Threshold**, except that instead of choosing the speed which is immediately above the current one, we rather increasing the speed by some parameterΔs.

This parameter should be carefully set: a too small increase of speed Δs will lead to a very small gap between the actual execution time of the task and the bottleneck communication or computation time so that in the event of a task failure, it will take many periods to catch up and no failure should hit the same task during that time. For instance, in Fig. 6, rectangles with rounded corners represent tasks running on different processors and other rectangles between them represent buffers. Only buffers in-use are depicted. datasets are labeled by colors. The vertical dashed lines indicate the start or end of a period. error and re-exe represent respectively an error happened and the re-execution afterwards. As we can see, an error strikes task 2, after two periods, it catches up.

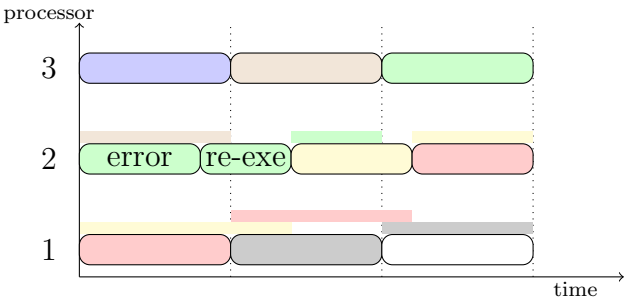


Figure 6: fault tolerance with buffers instance.

A.2 Heuristic CloserC for continuous speeds

Heuristic **CloserC** is the straightforward adaptation of **Closer** for the case of continuous speeds. In a first step, tasks speeds are set as s'_i so that $\frac{w_i}{s'_i} = P_{\mathrm{t}}$. However, in Line 8 of its counterpart, s_i is set to the speed s_i which exactly equals to $coef \times s'_i$ instead of the smallest discrete speed that is not smaller than $coef \times s'_i$.

A.3 Heuristic BestTradeC for continuous speeds

As we did in **BestTrade**, we still try to keep the best tradeoff between increase in probability and decrease in energy, but now cores can set speed exactly to tasks's critical speeds. First we set all tasks' speed to $s_c^i = w_i/(P_\mathrm{t} - w_i/P_\mathrm{t})$. Then, tasks with higher speeds (which are the one with the highest

Algorithm 7 BestTradeC(p, n)

We assume all weights are different ($w_i \neq w_j$ for $i \neq j$).
for $j = 1$ **to** $j = n$ **do**
  $s_j \leftarrow w_j/(P_\mathrm{t} - w_j/s_\mathrm{max})$, $m_j = 0$
end for
$i \leftarrow 0$, $S_{reduce} \leftarrow \emptyset$,
Sort tasks by non-increasing weight, such that $w_1 > w_2 > \ldots > w_n$
while $P(P_\mathrm{act} > P_\mathrm{t}) < proba_\mathrm{t}$ **do**
  $i \leftarrow i + 1$, $S_{fine} \leftarrow \emptyset$
  Put task T_i into S_{reduce}
  $s_c \leftarrow w_{i+1}/(P_\mathrm{t} - w_{i+1}/s_\mathrm{max})$
  $s_d \leftarrow w_i/P_\mathrm{t}$
  for all task T_j in S_{reduce} **do**
    if $\max(s_c, s_d) < w_j/P_\mathrm{t}$ **then**
      Remove T_j from S_{reduce}, put it in S_{fine}
      $s_j \leftarrow w_j/P_\mathrm{t}$
    else
      $s_j \leftarrow \max(s_c, s_d)$
    end if
  end for
end while
Compute s such that if $s_i = s$ for tasks in S_{reduce} and if $s_i = \max(s, w_i/P_\mathrm{t})$ for tasks in S_{fine}, then we have $P(P_\mathrm{act} > P_\mathrm{t}) = proba_\mathrm{t}$
for all task T_i in S_{reduce}, task T_j in S_{fine} **do**
  $s_i \leftarrow s, s_j \leftarrow \max(s, w_j/P_\mathrm{t})$
end for
$p_{av} \leftarrow p - n$; (Number of available processors)
for all task T_i **do**
  if $2w_i(\frac{w_i}{P_\mathrm{t}})^2 < w_i s_i^2 + f_i(s_i) w_i s_\mathrm{max}^2$ and $p_{av} > 0$ **then**
    duplicate T_i: $m_i \leftarrow 1$; $s_i \leftarrow w_i/p_i$; $p_{av} \leftarrow p_{av} - 1$
  end if
end for
return $< s_i, m_i >$ and S_{reduce}

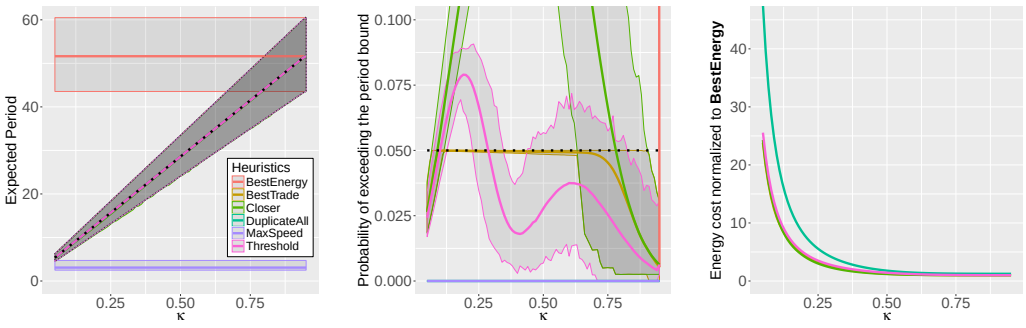
weights) are first slow down: we carefully decrease the speed of all faster tasks to the next critical speed. Whenever the reduction crosses the critical speed s_c^i of some task T_i, this task is included in the set of tasks currently being slowed (S_{excess}). We make sure that no task is assigned a speed smaller than its s_d^i: such tasks are removed from S_{excess} and put into S_{fine} , to remember that their speed cannot be reduced anymore. We stop when the target probability is exceeded: then, all the tasks that were still in S_{excess} are accelerated to reach the exact target probability. Finally, we deal with duplication as in the discrete case.

B Continuous-speed results

With the continuous-speed model, results are quite close to the results with the discrete-speed model. All heuristics meet the period bound and still only **BestTrade** meets the probability bound. Since they have more choices on speed, the difference in terms of energy consumption between **Threshold**, **Closer** and **BestTrade** is larger. **Closer** is the most energy saving heuristic on synthetic applications and **BestTrade** on real applications. All these three heuristics performance on energy is closer to **BestEnergy** than in the discrete case. Detailed results follow.

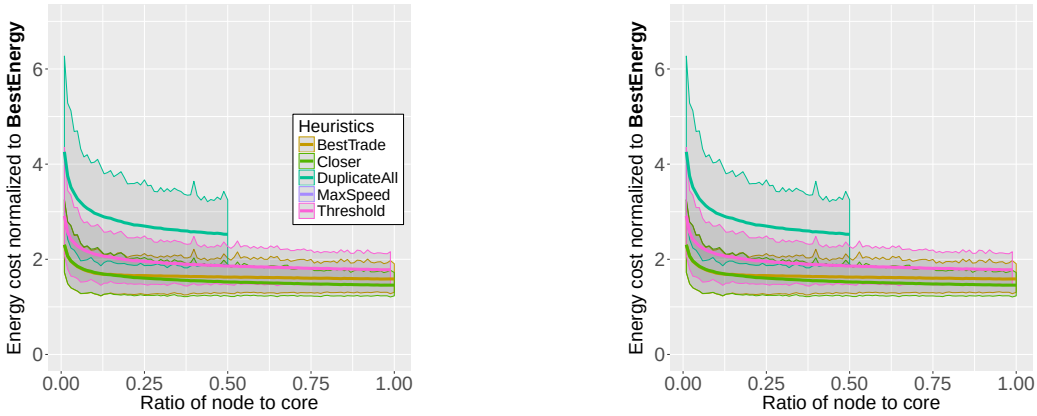
B.1 Synthetic applications and continuous speeds

Fig. 7a depicts all heuristics' performance on expected period by their best, mean and worst cases. Ratio of node to core here is 0.5. A larger κ represents a looser period bound. The dashed black line is the target period. **DuplicateAll** and **BestTrade** are covered by **Threshold**. All heuristics



(a) Expected period (b) Probability (c) Energy consumption

Figure 7: Energy consumption and constraints on synthetic applications, as a function of κ.



(a) Chain's energy cost

(b) Node's energy cost

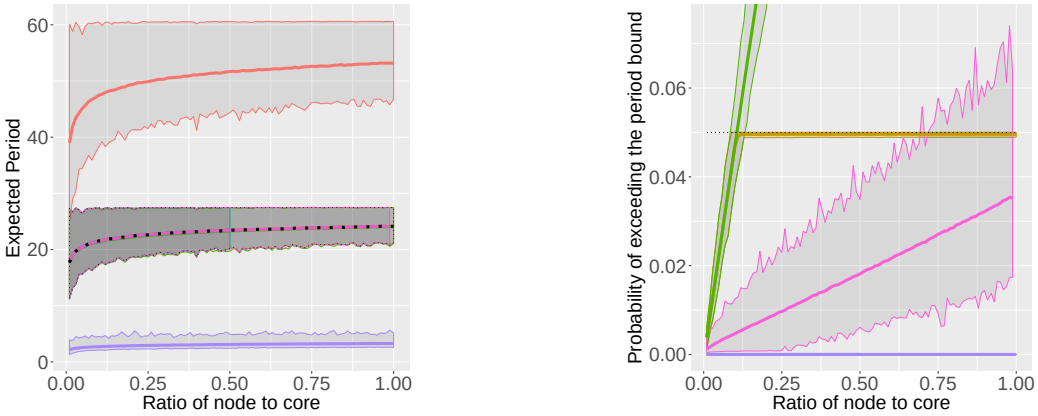
Figure 8: Energy consumption, as a function of chains' size.

except **BestEnergy** meet expected period bound, even **BestTrade** in this setting, which is not designed especially for bounding the expected period.

As shown in Fig. 7b, the target probability (depicted as dashed line) is set as 0.05, **DuplicateAll** is covered by **MaxSpeed**. Only **BestTrade** meets bounding the probability of exceeding with all target probability settings. Then **Threshold** sometimes can meet the target. Probability of **BestEnergy** is always 1 except when $k = 0.95$ some are 0. Fig. 7b also shows that when target period gets larger, it's much easier for all heuristics to meet the probability bound. Since when k gets larger than 0.5, P_{t} is relatively large, so few tasks are in set S_{excess}.

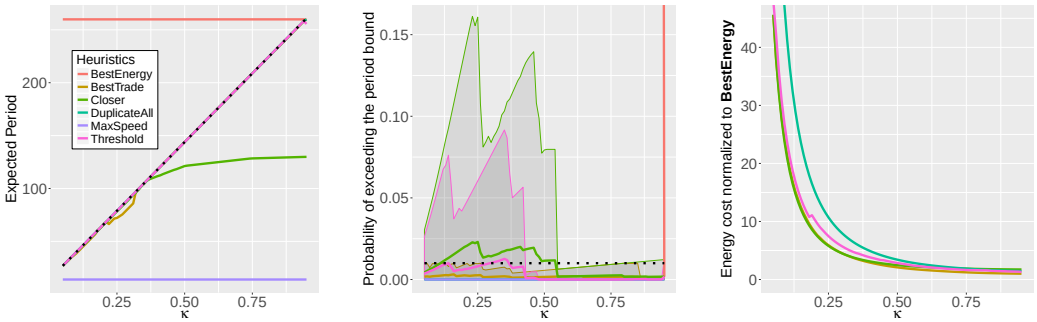
Fig. 7c presents heuristics' performance on energy saving. As **BestEnergy** is a baseline, all others are normalized to it. Except **BestEnergy**, the most energy saving heuristic is **Closer**, then closely followed by **BestTrade**, then **Threshold**. **MaxSpeed** is 215 times larger, so not included in the figure. Fig. 7c also demonstrates that given a larger period target, our heuristics' performance get close to **BestEnergy** and the difference between them is smaller. Then, we find size of chains also influences heuristic's performance. Fig. 8a and Fig. 8b illustrate heuristics' energy consumption on a chain or on a node respectively with varied sizes of chains. κ here is 0.4. Except for **MaxSpeed**, the more nodes a chain has, the closer to **BestEnergy** the heuristics' energy consumption becomes. As Fig. 8b shows, each node's energy consumption has the same tendency. The energy cost of heuristic **MaxSpeed** is too large, 215, so it is not included.

Fig. 9a illustrates heuristics' performance on expected period with different chain sizes. The dashed line is period bound. **BestTrade** is covered



(a) Expected period (b) Probability

Figure 9: Performance on constraints, as a function of chains' size.



(a) Expected period (b) Probability (c) Energy

Figure 10: Energy consumption and constraints on real applications, as a function of κ.

by other heuristics. Except **BestEnergy**, all others meet the target. For bounding the probability of exceeding the target period, see Fig. 9b: only **BestTrade** meets the target probability with all chain sizes, and **Threshold** is more likely to meet the target probability on small chains than large chains. **DuplicateAll** is covered by **MaxSpeed** in Fig. 9b. κ here is 0.4.

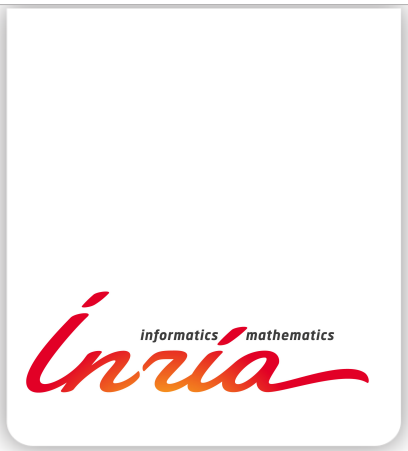
B.2 Real applications and continuous speeds

Fig. 10a shows heuristics' performance on expected period with varied period bounds. Except **BestEnergy**, all other heuristics meet the period target. For clarity, only the mean is depicted, but the maximum of heuristics' expected period also do not exceed the period bound. **BestTrade** is partially

covered by **Threshold** and **DuplicateAll** is covered by **Threshold**.

Fig. 10b demonstrates that only **BestTrade** can meet the probability target with all period target settings. The results of **BestEnergy** are always 1 except some are 0 when $\kappa = 0.95$. With a larger period target, it is easier to meet the probability target.

Different to results on synthetic applications, Fig. 10c shows that when the target period becomes large ($k \geq 0.375$), **BestTrade** becomes the most energy-saving heuristic instead of **Closer**. **MaxSpeed**'s energy consumption is too large, 254, so not included in this figure. It demonstrates that **BestTrade** can be the most energy-saving heuristic in some cases.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée

655 avenue de l'Europe Montbonnot

38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399